opsi manual opsi version 4.1

uib gmbh
# Table of Contents

1. Copyright .................................................................................................................. 1

2. Introduction ............................................................................................................... 2
   2.1. Who should read this manual? ........................................................................... 2
   2.2. Conventions in text and graphics .................................................................... 2

3. Overview of opsi ....................................................................................................... 3
   3.1. Experience ......................................................................................................... 3
   3.2. opsi features ..................................................................................................... 3
   3.3. opsi Extensions ................................................................................................. 3
   3.4. Structure ............................................................................................................ 4

4. opsi-Management GUI: opsi-configed ................................................................. 6
   4.1. Requirements and operation ............................................................................ 6
      4.1.1. Logging of the opsi-configed .................................................................. 7
      4.1.2. Choosing the language .......................................................................... 8
      4.1.3. Logging of the ....................................................................................... 9
   4.2. Login .................................................................................................................. 9
   4.3. Copy & Paste, Drag & Drop ............................................................................ 10
   4.4. opsi-configed modes Client configuration / server configuration / license management .......................................................................................................................... 11
   4.5. Depot selection ................................................................................................. 11
   4.6. Client selection ................................................................................................. 12
      4.6.1. The clients list ......................................................................................... 15
      4.6.2. Selecting clients ....................................................................................... 16
   4.7. Client selection and hierarchical groups using the tree view ....................... 19
      4.7.1. Basic concepts ......................................................................................... 19
      4.7.2. How to .................................................................................................... 19
   4.8. Client processing / Client actions .................................................................. 20
      4.8.1. Install By Shutdown, Uefi Boot and WAN Configuration ...................... 21
      4.8.2. WakeOnLan (Wake selected clients) ...................................................... 22
      4.8.3. Fire opsiclientd event (Push Installation) ............................................. 23
      4.8.4. Sending messages (Show popup message) ........................................... 23
      4.8.5. Session info for selected clients ............................................................. 24
      4.8.6. For WAN-Clients: Delete package cache ............................................. 24
      4.8.7. Call external remote control tools for selected clients ........................... 24
      4.8.8. Shutdown / reboot of selected clients .................................................... 26
      4.8.9. Delete, create, rename and move clients .............................................. 26
   4.9. Product configuration ....................................................................................... 28
   4.10. Property tables with list editor windows .................................................... 30
      4.10.1. Hidden Password Property Values ...................................................... 31
7. Security
   7.1. Introduction
   7.2. Stay tuned
   7.3. General server security
   7.4. Client authentication at the server
   7.5. Server authentication at the client
      7.5.1. Variant 1: verify_server_cert
      7.5.2. Variant 2: verify_server_cert_by_ca
   7.6. Authentication at the control server of the client
   7.7. Admin network configuration
   7.8. The user pcpatch
   7.9. Webservice access limitations
   7.10. Change the bootimage root password

8. opsi products
   8.1. Localboot products: automatic software distribution with opsi
      8.1.1. opsi standard products
      8.1.2. Manipulating the installation sequence by priorities and dependencies
      8.1.3. Integration of new software packages into the opsi software deployment.
      8.1.4. Subscription
      8.1.5. Configuration of default properties
   8.2. Netboot products
      8.2.1. Parameters for the opsi linux boot image
      8.2.2. Unattended automated OS installation
      8.2.3. Some hints to the NT6 netboot products (Win7 to Win 10)
      8.2.4. memtest
      8.2.5. hwinvent
      8.2.6. wipedisk
   8.3. Inventory
      8.3.1. Hardware Inventory
      8.3.2. Software Inventory
   8.4. opsi subscriptions
      8.4.1. Initial Deployment of opsi subscriptions
      8.4.2. Configuration of the `opsi-package-updater` / updating the opsi subscriptions
      8.4.3. Configuration of default properties
      8.4.4. Subscription `MS-Hotfixes`
      8.4.5. Update subscription for `MS-Office Hotfixes`
      8.4.6. Update subscription for the opsi standard packages

9. opsi Extensions
   9.1. Activation of non-free modules
   9.2. User roles (via opsi-configed)
9.10. opsi License Management
9.10.1. Conditions for using the opsi License Management extension
9.10.2. Overview
9.10.3. License-pools
9.10.4. Setting up licenses
9.10.5. Editing licenses
9.10.6. Assignment and release of licenses
9.10.7. opsi with secureboot
9.10.8. The opsi-vhd products
9.10.9. Introduction
1. Copyright

The Copyright of this manual is held by uib gmbh in Mainz, Germany.

This manual is published under the creative commons license Attribution - ShareAlike (by-sa).

A description of the license can be found here: https://creativecommons.org/licenses/by-sa/3.0/

The legally binding text of the license can be found here: https://creativecommons.org/licenses/by-sa/3.0/legalcode

Most parts of the opsi software is open source. Not open source are the parts of the source code which contain new extensions, that are still under cofunding, which have not been paid off yet. See also: opsi cofunding projects

All of the open source code is published under the AGPLv3.

The legally binding text of the AGPLv3 license can be found here: http://www.gnu.org/licenses/agpl-3.0-standalone.html

Information about the AGPL: http://www.gnu.org/licenses/agpl-3.0.en.html

For licenses to use opsi in the context of closed source software, please contact uib gmbh.

The names opsi, opsi.org, open pc server integration and the opsi logo are registered trademarks of uib gmbh.
2. Introduction

2.1. Who should read this manual?

This manual is aimed at everyone who is interested in automatic software distribution with opsi. The focus of the documentation is the explanation of the technical background, in order to contribute to an understanding of the processes.

This manual should not only support the system administrator who works practically with opsi, but also give prospective users a concrete overview of opsi in advance.

2.2. Conventions in text and graphics

Names which are shown in <angle brackets> have to be replaced by a real name.

Example: The file share on which the opsi software packages are located is called <opsi-depot-share> and is located on a real server on /var/lib/opsi/depot.

The software package: <opsi-depot-share>/ooffice is actually under /var/lib/opsi/depot/ooffice.

Examples from program code or configuration files use a Courier font and are highlighted in color.

depoturl=smb://smbhost/sharename/path
3. Overview of opsi

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

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

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

3.1. Experience

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

You can find an geographical overview of the registered opsi-installations at: opsi-map

3.2. opsi features

The core features of opsi are:

- automatic software distribution
- automatic operating system installation
- hard- and software inventory
- comfortable control via the opsi management interface
- support of multiple depot-servers

3.3. opsi Extensions

- Management of licenses
- MySQL-Backend
• Nagios Connector
• Installation ab Shutdown
• Local Image Backup (Rapid client restore of student computers. For public authorities (e.g. schools) only)
• Linux Agent
• WAN Extension (Support for clients behind slow connections)
• User Profile Management (manipulation Profiles even with Roamin Profiles)
• OTRS::ITSM Connection - via KIX4OTRS by cape IT gmbh

3.4. Structure

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

opsi supports different backends: Backends:

• File based
• MySQL based

![Diagram showing the structure of opsi](image)

Figure 1. Scheme: opsi with File / MySQL backend

More details you will find at Section 5.6, “opsi data storage (backends)”.

The backend configuration will be found at the files in the directories `/etc/opsi/backendManager` and `/etc/opsi/backends`. 
The configuration files in `/etc/opsi/backends` define the backends.

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

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

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

A more detailed reference of these configuration files you will find at Section 5.7.1, “Common configuration files in /etc”.
4. opsi-Management GUI: *opsi-configed*

### 4.1. Requirements and operation

Since version 4.0.7.5.22, the *opsi-configed* requires at least Java 1.8. It works with the data from a running *opsiconfd* at least of version 4.0.6.

In non-opsi systems the *opsi-configed* can be simply installed by copying the required jar archive files as contained e.g. in the opsi-package. In order to automate this, you find will two helpers at [https://download.uib.de/opsi4.1/misc/helper/](https://download.uib.de/opsi4.1/misc/helper/): the setup program *opsi-configed-setup.exe* works for Microsoft Windows and *opsi-configed-linux-setup.tar.gz* works for various Linux systems.

If the client has a graphical desktop *opsi-configed* can be started from a start menu entry.

On Linux it can be started in a shell by executing `/usr/bin/opsi-configed`.

With non-gui servers there exist only the special non-gui start options:

- `--version`
- `--help` bzw. `-h`
- `--querysavedsearch [SAVEDSEARCH_NAME]` bzw. `-qs [SAVEDSEARCH_NAME]`
- `--swaudit-pdf FILE_WITH_CLIENT_IDS_EACH_IN_A_LINE [OUTPUT_PATH]`

If the additional required jar archives exist in the path the *opsi-configed* can be started simply by `java -jar configed.jar`.

With `java -jar configed.jar --help` you'll get a list of the command line options.
4.1.1. Logging of the opsi-configed

By default, the opsi-configed uses its log level 3 "Info". The level can be raised to 4 "Check" or 5 "Debug".

To change the log level the command line option --loglevel [LEVEL] can be used. It is not recommended to set level 5 as long as not the start process needs to be inspected. For, with level 5, the produced log file is very large; it is difficult to get loaded and viewed. When the opsi-configed is running and a potential error situation is expected the log level can be raised via the menu entry Help/ConfigEditor log level.
Level 4 can be helpful since with it, the service calls are logged. With luck, level 5 offers a detailed view of actions.

Since version 4.0.7.6.12 the logfiles are deposed by default in the user home directory. In Windows the folder

c:\Users\[User name]\AppData\Roaming\opsi.org\log

In Linux the default logfiles folder is the (hidden) subfolder ".configed" in the user home directory.

The current logfile is named configed.log, the up to 3 preceding versions are configed_0.log, configed_1.log, configed_2.log.

The logging directory can be changed via the command line option "-d".

The current logfile path is displayed at the Help menu, entry "Current logfile". The filename can there be retrieved in order to use it in an open file dialog of a viewer program or can be directly opened by the default application for .log files.

4.1.2. Choosing the language

The `opsi-configed` tries to use the language following the OS defined locale. If the matching translation file is missing English is used as default language. If terms in translations file are missing the expressions of the Englisch translation are used as default.

When calling the `opsi-configed` you can set a locale via the command line option

-`-l` resp. `--locale`

On principle, the locale has the format `language_region`, each component with two characters, eg. `en_US` of `de_DE`. It suffices to give the two character language code since there no region specific variants prepared.

In a running `opsi-configed` the language can be switched via the menu item File/International. A change triggers a re-initialization of the program with a (nearly complete) rebuilding of the visual components in the new language.

Finally the call parameter

--`localizationfile`

can be used for directly prescribing a localization file. The additional parameter

--`localizationstrings`

has the effect that the display of the localized expressions is combined with displaying the terms for which expressions should be given. This can be used for producing and testing a localization file.
4.1.3. Logging of the opsi-configed

By default, the opsi-configed uses its log level 3 "Info". The level can be raised to 4 "check" or 5 "debug".

To change the log level the command line option --loglevel [LEVEL] can be used. It is not recommended to set level 5 if not already the start process has problems since, with level 5, the produced log file is very large; it is difficult to view. When the opsi-configed is running and a potential error situation is expected the log level can be raised via the menu entry Help/ConfigEditor log level.

Level 4 can be helpful since with it, the service calls are logged. Level 5 often offers a detailed view of actions.

By default the logfiles are deposed in the user home directory in the subfolder ".configured". The current logfile is named configed.log, the up to 3 preceding versions are configed_0.log, configed_1.log, configed_2.log.

In newer versions of windows the user home directory is interpreted as

c:\users\[USERNAME]\appdata\local

Observe that the Windows Explorer shows a localized version of "users" and does not give "AppData" in the listing of "c:\users\[USERNAME]\". But the AppData folder contents appear, when "appdata" is supplemented to its superfolder name.

The logging directory can be changed via the command line option "-d". The opsi product opsi-configed uses this option to set the logging directory to

c:\opsi.org\log

But only local administrators can write to this location. If other users start the opsi-configed the location

c:\users\[USERNAME]\appdata\local

will therefore again be used.

4.2. Login
The `opsi-configed` tries to connect to the opsi server via https. The login is done with the given parameters `opsi server[:Port]` (default port 4447 – opsiconfd) and the user/password pair of the `opsi-config-server` account. For a successful login the provided user has to be a member of the unix group `opsiadmin`.

In the local user profile, the `opsi-configed` saves certain session info in order to rebuild the essential working context after a restart, in particular a selected client group. Since version 4.0.7 the session data is used to produce a selection list of opsi servers to which you were connected (e.g. a productive one and a second one for experimental purposes). The last server used gets the highest place, and can be directly used again.

The gzip compression in HTTP protocol reduces the amount of data being transferred at the expense of an extended processing time, this is due to the fact that the data must be compressed and uncompressed. It has been observed that the reaction times tend to be shorter without compression in the local network, as the effects normally surpass the prolonged processing time. For transmissions over the WAN, it tends to be the opposite. In the practice, little difference is noticed on LAN connections, but relevant differences are noticed on WAN connections, so the Gzip option is enabled by default.

The feature `check which clients are reachable` runs in the background and shows the results in the client table. It can be enabled from the login screen mask or via command line parameter. The default refresh interval is 0 min (= deactivated). It should be observed though that a too short refresh interval produces a lot of network waiting states which can slow down the opsi server.

4.3. Copy & Paste, Drag & Drop

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

### 4.4. `opsi-configed` modes Client configuration / server configuration / license management

To switch between the different usage modes of the `opsi-configed`, use the buttons in the upper right corner of the `opsi-configed` frame. Since version 4.0.4, there are six buttons.

![Figure 4. `opsi-configed`: Usage modes](image)

The first three buttons allow you to change the editing target of the main window: client configuration, server configuration. On the other hand, each of the buttons *group actions*, *product actions* and *license management* starts a special window to manage the specific objects or actions.

These windows can as well be opened via the main menu item *windows* (since `opsi-configed` version 4.0.7).

### 4.5. Depot selection

All `opsi-depot` servers that are integrated with your server, are listed in the upper left corner of the `opsi-configed`. By default the depot on your `opsi-config-server` is selected and the clients belonging to this `opsi-depot` are shown.

You can select multiple Depots at the same time and edit their clients together. However, only the selected depots are synchronized with each other. Trying to edit clients from asynchronous depots together will be rejected with an appropriate warning and the corresponding error message.

As of version 4.0.5, there is no need to carry out a complete data-reload when switching to a different depot-server, that means, when you select a depot its data is loaded immediately. In addition, there are the following buttons:

- `(=+)`: Marks all depots with identical product stocks.
- `(++)`: Marks all depots (you can also use `Ctrl-a`)
4.6. Client selection

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

Since version 4.0.4, the opsi-configed saves on the local machine, for the current user, the current depot server and group selection. If the opsi-configed is restarted, you can continue working at the point where you were.

Please note, that group selection is preserved when changing depot selection. In order to see all clients in the other depot the group selection has to be changed appropriately. the same pattern of work.
Figure 6. opsi-configed: client selection mask

You may select a line of the list not only by manual scrolling and selecting but also by a String search. This requires that you enter a String into the search field at the top of the list.

How the search works is determined by the selected elements in two drop down lists:

Via field selection you can choose if

- all fields (more precisely, all fields that are for this temporary configuration represented as columns) are searched (default), or
- only one field (and which one) is searched.

Concerning the method of search you may select between the options (since 4.0.7):

- Full-text: the search string is used in a similar way to a web search on a certain search engine for example in the standard manner; i.e., if the input contains several keywords (delimited by blanks) then the word elements will be a match if any of the input parts are fully contained in some of the columns.

- Full-text (complete string): the search string is used like using a web search on a search engine the search string embraced by citation marks; i.e. a table line will match if the complete input string is part of one of the columns content.

- Start-text search: a table line will be a match if the column text starts with the search string.

- Regular expression: the search string is interpreted as a so called regular expression; i.e., a line will be a match if the input string produces a match according to the rules of regular expressions.
(as described in the java doc for java.util.regex.Pattern).

The enter key produces the next search hit. If there is no match it advances the mark to next line.

More selection functions based on String search are shown in the context menu of the search field.

![Figure 7. opsi-configed: Search function in the client selection list](image)

**Examples of Search Patterns**

All PCs which have a name or a description containing the character sequence *Miller* with capital *M* or with *m* are found by using the pattern

```
.*iller. *
```

The dot in ".*" means "arbitrary character", the asterisk " * " means "arbitrary number of occurrences (of the beforehand designated element)". That is

```
.*iller. *
```

it matches, if anything (any number of any characters) come before *iller* and anything after *iller*. Since "any number" may be zero

**Home of Miller**

matches where no character follows after *iller*.

But to ensure that we do not mark *Tiller* as correct a more precise pattern would be

```
.*[Mm]iller. *
```

Several characters enclosed in brackets are interpreted as the searched value must contain *one of the enumerated characters*. With this more precise pattern, every string is recognized which contains either *Miller* oder *miller* but no other string.

Here is yet another example, a pattern search for products:

```
0.-opsi.*standard
```

matches for all products which have a name beginning with "0", followed by an *arbitrary character*, followed by -*opsi*, followed by *arbitrary characters* (in *arbitrary number*); finished by *standard*. 

To ensure that the second character is a number symbol, i.e. one of the characters "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", one can write

0[0123456789]-opsi.*standard

As short hand for [0123456789] one can use, since it is a complete partial sequence of the sequence of all characters, [0-9]. Therefore the search pattern reduces to

0[0-9]-opsi.*standard

Matching products e.g.

03-opsi-abo-standard

or

05_opsi-linux_standard

More informations on the topic can be found in the java api doc, key word "java.util.regex.Pattern".

4.6.1. The clients list

The clients list has per default the columns client name, description, on, IP address 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 top right hand side of the window
- On shows after clicking the button Check which clients are connected the result of this query. This feature runs in the background and shows the results in the client table. It can be enabled from the login screen mask or via command line parameter. The default refresh interval is 0 min (= deactivated).

Figure 8. opsi-configed: Button Check which clients are connected

Figure 9. opsi-configed: Client reachable

- IP address shows the IP number to which the opsi server resolves the client name.
- last seen shows the date and a time of the last client connect to the opsiconfd web service

Some columns are deactivated by default:
• **session infos** (data is retrieved from the operating system running on the specific client)
• **Inventory No** (displaying some optionally entered data)
• **created** (date and time of client creation)
• **opsi mac address** (hardware address of the client as used by opsi)

You may activate these columns using the context menu. The configuration of the columns being displayed may be changed using the entry `configed.host_displayfields` in the server configuration.

![Figure 11. opsi-configed: change the default for visible columns in the clients list](image)

Adding the column **session infos** enables the button "request session infos from all clients" in the button panel.

![Figure 12. opsi-configed: Button Sessioninfo](image)

When this button is clicked the `opsiconfd` tries to connect to all clients and to retrieve data of the active user sessions. From the result, the account names are shown in the column **session infos**. Instead of using the button you may start the request only for the selected clients via the context menu or the main menu entry **OpsiClient**. By doing this, network timeouts are avoided.

Since the search function for the client list works (if not configured otherwise) on all displayed columns you may now find out which is the client belonging to a logged in user (with known account name).

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

### 4.6.2. Selecting clients

You can select one or multiple clients to work with. The client view can be restricted to the selected clients by clicking the funnel icon or from the menu by **Grouping / Show only selected clients**.
A selected client group can be saved with the icon Save grouping or from the menu by Grouping / save group with a free selectable name.

You can use the mouse to add the selected clients to an existing group (by dragging them to an existing group which is displayed in the tree view).

In the client selection dialog (as called via menu Selection / Define search) clients can be selected using a variety of criteria based on their configurations.

E.g., it is possible to search for opsi installed products as well as software found by the opsi software audit. You may as well search for PCs satisfying certain hardware conditions. Criteria may be combined by logical AND or OR operations and may be negated by a NOT (which is produced by a click on the Not-Field before the property field). Search strings can be given as fixed strings combined with asterisks * as wildcard symbols.

Search definitions can be saved and then again used via the menu item Selection/Use saved search definition.
It is also possible to run a saved search from the command line when the opsi-configed editor is started. By including the flag "-qs" and the name of the saved search, the configuration editor will start with the saved search results. If the name is omitted, then a list of available searches will be displayed.

To detect failed installations, the menu item Selection offers Failures with product and Failures occurred (today, since yesterday, ...), since version 4.0.5. Choose the first setting to get a list of all products. If you select a single product, all clients will be shown, where the installation of this product failed. When choosing for instance Failures occurred - today, all the clients will be marked, where an product installation failed today.
4.7. Client selection and hierarchical groups using the tree view

Clients can be grouped in a convenient way by using the tree view control placed on the left side of opsi-configed frame.

4.7.1. Basic concepts

The tree view control has three base nodes groups, directory and client list. Alls clients of the selected depots are displayed in the group client list.

The nodes groups and directory are different in so far as each client can have any number of locations in the groups subtree but has a unique location in the directory subtree; as long as there was no other assignment to subgroup of directory a client is automatically placed into the group NOT_ASSIGNED.

If you select a client, all groups to which the client belongs will get color marked icons.

![Figure 16. opsi-configed: Tree view with clients and groups](image)

4.7.2. How to ... 

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

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

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

You may also create groups by using the context menu above ALL or any existing group.
You will be asked for the new group's name.

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

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

A group can

- be moved to a different location via drag & drop.

The context menu of a group item can be used

- to create a subgroup;
- to edit the group properties;
- to delete the group together with its subgroups and all client assignments of them;
- to remove all client assignments while keeping the group and its subgroups;
- to display the the contained clients and select them in one step.

4.8. Client processing / Client actions

Using the menu OpsiClient or the context menu in the Clients tab you may choose from a lot of client
4.8.1. Install By Shutdown, Uefi Boot and WAN Configuration

Several client standard configurations can be applied directly in the client information panel which is located on the right side of the clients page. Please observe that UEFI support and WAN configuration both are currently based on non free extension modules. If these modules are not active the corresponding buttons are disabled.

- Install By Shutdown:
  In Section 9.19, “opsi Installation on Shutdown (free)” , the method do switch to Install by Shutdown installation is described in detail. You can automatically trigger this configuration by pushing the On-Button for InstallByShutdown. Observe that this requires a opsi-client-agent reinstall or reconfigure. Alas, the state of the configuration can currently only be seen in the product property `on_shutdown_install` of the opsi-client-agent product.

- Uefi Boot:
  The state of checkbox Uefi Boot indicates, if a client is configured for UEFI boot. It is activated the value of the client host parameter `clientconfig.dhcpd.filename` is changed to `linux/pxelinux.cfg/elilo.efi` geändert. (For more infos cf Section 9.6, “opsi with UEFI / GPT”)

- WAN configuration:
  The opsi configed checks, if the standard Wan configuration is available or not. The check mark
for the WAN configuration will be set or not accordingly. If you check the box manually, the default configuration will be set.

This configuration is no longer hard coded since version 4.0.7.6.5, but is read from the server host parameters configed.meta_config.wan_mode_off* and is also interpreted as a negation. If you have kept the automatically set Meta_Config.wan_mode* parameters, the one described in Section 9.11.5, “Recommended configuration when using the WAN/VPN extension module”, can be taken as an example or recommendation for a standard WAN configuration given by uib gmbh.

Whether the client is configured as a WAN client or perform an UEFI boot, can also be displayed in the appropriate columns on the client table. For the current session you can find on the menu item OpsiClient or activated in the context menu, permanently via the entry configed.host_displayfields for the server host parameters. This can be seen directly in the overview, for which clients the property is set, and it can also be searched for and sorted out.

![Figure 20. opsi-configed: Erweiterte Spaltensicht für opsi-Clients](image)

### 4.8.2. WakeOnLan (Wake selected clients)

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

Since version 4.0.7 you can choose

- if the network signal is meant to be sent to the selected clients at once
- which delay should be between the waking of two clients
- when the process shall start (via a scheduler).

If a client is assigned to a depot server which is not the configserver then the Wake On Lan signal is not directly sent to the client, but the opsi-configed tries to establish a HTTPS connection to the opsiconfd of the depot server which in turn sends the Wake On Lan package to the client inside its network segment.
It should be observed that it is the `opsi-configed` which triggers the actions, therefore the program must not be shut down in the meantime.

### 4.8.3. Fire `opsiclientd` event (Push Installation)

This menu entry is used to send to the `opsi-client-agent` on the selected clients a command to fire the event which is selected in the submenu. The standard event is "on demand" which means the demanded action is started at once. Be aware that this may have the effect that the client is rebooting without any warning.

To incorporate additional events (which should be configured in the `opsiclientd.conf`) into the submenu you have to edit the config `configed.opsiclientd_events` via the tab (server) host parameters.

All messages will be shown on the active desktop. If the client isn’t reachable, you will get a message.

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

### 4.8.4. Sending messages (Show popup message)

Choosing the menu entry `Show popup message` you will get a small edit window where you can type in your message.
By clicking on the red tick you will send the message to the selected clients.

At the selected clients a message window will appear.

4.8.5. Session info for selected clients

The selected clients get the signal to communicate their session information. The data is shown in the session info column (if visible).

4.8.6. For WAN-Clients: Delete package cache

On WAN clients there are occasional problems with the package cache synchronization. This function resets the cache.

4.8.7. Call external remote control tools for selected clients

The option *Remote Control Software call* in the client context menu as well as the client main menu (since *opsi-configed* version 4.0.1.11) is very powerful. It can be used to use any command that the operating system offers, parametrized e.g. by the client name.

As an example there are configurations automatically generated which can be used to send a ping to the selected client: one ping command that works in Windows environment and one command that requires a Linux X environment. Please observe: *opsi-configed* calls obviously the command in its
environment, i.e., we need the Linux command when the *opsi-configed* is running in Linux.

![Image](image_url)

**Figure 24. opsi-configed: Choice of Remote Control call**

The selection window has three parts. The upper part lists the names of the existing commands. It follows a line, which shows the selected command and offers the chance to edit it (if this is allowed). Additionally, the line contains the buttons to execute or abandon the action. The third text area of the window captures any messages that are returned by the operating system when calling the command.

These calls offer a quasi infinite range of opportunities. For example, a command can be configured to open a Remote Desktop connection to the selected client (if it allows such connections). On a Windows system, such a command is

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

In a Linux environment the following command can be used:

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

In these examples serves `%host%` as a variable, which *opsi-configed* automatically replaces by the value for the selected host. Other variables that can be analogously used in the commands are:

- `%ipaddress%`
- `%hardwareaddress%`
- `%opsihostkey%`
- `%inventorynumber%`
- `%depotid%`
- `%configserverid%`

If the command is marked by the additional server configuration entry *editable* as `true`, then the command line allows ad hoc editing. For example, you may supply a requested password or vary the command as needed.

If there is some command declared as *editable* then in fact any program addressed at the client computer can be called by changing the editable command.
If more than one client is selected the command will be executed in a own thread for each client.

The list of remote control commands is editable via server configuration entries (cf. Section 4.16, “Host parameters in client and server configuration”).

To define a command example, at minimum an entry `configured.remote_control.example` (or `configured.remote_control.example.command`) must be generated. The value of property has to be the command (in which the variables `%host%`, `%ipaddress%` etc. can be used). Additionally, an entry `configured.remote_control.example.description` can be defined. The value of this entry will be shown as tooltip (if not existing, the command itself will serve as tooltip content). Furthermore, a Boolean entry `configured.remote_control.example.editable` can be added. If its value is set to `false` the command cannot be edited in the selection window.

![Figure 25. opsi-configed: Editing of remote control commands in the server properties editor](image)

### 4.8.8. Shutdown / reboot of selected clients

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

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

### 4.8.9. Delete, create, rename and move clients

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

If you choose to create a client, an input mask opens. There you enter or confirm the required data – client name without domain specification, domain name, depot server name. You may add a textual description for this client and notes on this client.
The mask also contains fields for an optional declaration of the IP-number and the ethernet (MAC) address of a client. If the backend is activated for the configuration of a local dhcp-server (which is not the default setting), this information will be used to make the new client known to the dhcp-server. Otherwise the MAC address will be saved in the backend and the IP-number will be discarded.

When creating clients you can directly for the new client specify to which group it should belong, as well as which netboot product should be directly set on setup. In addition, you can activate directly the Install by shutdown, UEFI Boot and the (standard) WAN configuration from the beginning. These settings can easily be made in the Hosts-List. These configurations are only available since the version 4.0.5.8.1.

Since opsi 4.0.4 it is possible to disable the options for creation and deletion of an opsi client. This is used if the client creation should be managed by a different service, eg. the UCS service.

For the configuration of these options, a host parameter (config) is provided. It is named config.host_actions_disabled and offers the list values

- add client
- remove client

(multiple selection allowed). The default is the empty selection meaning that no option is disabled.

The default setting can be changed so that adding and removing clients from the opsi-configed is disabled:
You may rename a selected client, you will be asked for the new name.

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

![Image](opsi-configed_change_depot.png)

**Figure 27. opsi-configed: change the depot of a client**

### 4.9. Product configuration

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

![Image](opsi-configed_product_configuration.png)

**Figure 28. opsi-configed: product configuration mask**
Since opsi 4.0.4 a search function is added.

With the search function, products can be searched by product names and (if desired) in combination with special values in the fields of the product table (like searching the client table). Therefore a search string can be entered. The search starts immediately and the first matching line is marked. If there is no match to be found (or characters are removed from the search string), the first line of the table is marked.

The context menu offers some more options.

![Figure 29. opsi-configed: Product search with context menu](image)

To get a better overview, activating the filter function reduces the product view to the selected products only. The selections stays active until the filter is disabled by clicking the filter button again.

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

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

This are the columns:

- **Status** is the last announced state of the product and can hold the values *installed*, *not_installed*, *unknown*. The table shows an empty cell if the value is *not_installed* to improve the usability of the view. The cell becomes grey if a multitude of selected clients is selected and does not share a common value (grey coloring represents the pseudo value *mixed*).

- **Report** informs about the progress or the result of the last action using the pattern `<action result> (<last action>)`. During an installation process there may be indicated *installing*, afterward e.g. *failed(setup)* or *success (uninstall)*.

- The column **Requested action** holds the information which action is to be executed. Possible values are *none* (shown by an empty cell) and the action types for which scripts are defined in the product package (possible values are *setup*, *uninstall*, *update*, *once*, *always*, *custom*).

- The field **Version** displays the software version number combined with the opsi package number of the software package installed on the client.

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

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

- The *position* column displays the product ordering forecast for installation sequences.

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

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

### 4.10. Property tables with list editor windows

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

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

![Figure 30. opsi-configed: property table with tooltip](image)

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

![Figure 31. opsi-configed: list editor, selection list](image)

Clicking a new value changes the selection.

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

![Figure 32. opsi-configed: list editor, edit field](image)

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

As soon as the edit field contains a new value – not yet occurring in the value list – the plus button will be activated with it the new value can be added to the list of values.

If multiple values are allowed – as it should be e.g. for the property *additional drivers* – a value may be added to the set of selected values by Ctrl-Click. The very same action removes a selected value from the set. The minus button empties the selection set completely.

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

### 4.10.1. Hidden Password Property Values

A property value that is a password should not be directly displayed.

Until this feature will be constructed as a special value type in some coming release the hack is used that a property value will only be displayed if the user explicitly requests it in cases (since version
4.0.7):

- the property key text contains the string *password*
- the property key text starts with the string *secret*

E.g., the value of the property *root_password* in the Linux netboot products is displayed as a sequence of stars (until the user does the edit click and explicitly confirms to showing the password).

### 4.11. Netboot products

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

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

![Figure 33. opsi-configed: mask to start the bootimage](image)

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

### 4.12. Hardware information

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

To simplify and automate the drivers of special clients and to upload them on the *(opsi-depot-server)*, since version 4.0.5, the option to select the paths from the hardware information is possible, thus the opsi-configed via the Share delivers the above mentioned. The two offered byAudit driver paths are composed of the manufacturer and the product or the model, which are respectively read from the computer and the mainboard. By clicking the right button to upload a driver, a new window will be displayed to add more settings.
If you open the opsi-configed on a Linux system, it is not directly possible to carry out a driver upload because the connection is carried out via a Share. This needs to be made manually. However, the methods or directory structures are an essential aspect of the drivers integration for Linux users as well as for Windows users.

Without further settings, the driver upload of a Windows computer, works only if the connection to the Share is enabled.

Among other things, information must be given in a new window, like to which Windows product should the driver be prepared, which drivers are to be uploaded and with which method or the directory in which the driver integration takes place. The target directory is accordingly changed with the selection of another method. The previously selected byAudit driver path can be found again by default in the selected method byAudit, that specifically integrates the selected driver for the type of machine.

Following methods and directories are possible:

- **standard**: For the drivers which are found in ./drivers/drivers/, the driver will be matched to the corresponding hardware using the PCI IDs (i.e. USB- or HD_Audio-ID) in the description file, and then integrated into the Windows setup as needed. It may be the case that the drivers found by opsi in this location do not necessarily work with your hardware. For the drivers which are found in ./drivers/drivers/, the driver will be matched to the corresponding hardware using the PCI IDs (i.e. USB- or HD_Audio-ID) in the description file, and then integrated into the Windows setup as needed. This is the fall back directory for all clients.

- **preferred**: In the case that you have to support special hardware, and you can find the additional drivers from the manufacturers, then use the following procedure to include them in the installation. Place the additional drivers in their own directory under: ./drivers/drivers/preferred (the naming and depth of the directory structure is not important).
Drivers that are found in the directory `.drivers/drivers/preferred` will be integrated into the Windows setup, assuming that opsi finds a suitable match to the drive hardware based off of the PCI IDs (i.e. USB or HD_Audo-ID) in the description file. Problems can occur when the same PCI ID of the drivers is found in `preferred`. In this case, a direct mapping of the drivers to the devices is needed.

- **excluded**: It could happen that the manufacturers include different drivers for different operating systems (i.e. Vista vs. Win7) or different configurations (i.e. SATA vs. SATA RAID). The `create_driver_links.py` cannot make this distinction. If you think the wrong driver has been installed, then move the driver to the `drivers/exclude` directory and then call `create_driver_links.py` again. Drivers in the directory `drivers/exclude` are not used during the integration.

- **additional**: When installing additional drivers based on the PCI-IDs or USB-IDs, they should be installed under the directory `.drivers/drivers/additional` (where name and depth of the directory structure is not important). You can map one or more drivers to a client using the Product-Property `additional_drivers` and a list of driver directories under `.drivers/drivers/additional`. The directories specified by `additional_drivers` are searched recursively until all drivers are found. This method can be used to make a specific directory based on the client type (i.e. dell-optiplex-815).

- **byAudit**: The previously described mechanisms that directly map drivers to devices is automated since the 4.0.2 Release 2 of opsi. Opsi will search the directory `.drivers/drivers/additional/byAudit` for a director name that matches the field `Vendor` that was given in the Hardware Inventory. This `Vendor` directory will be search for a `Model` directory that corresponds to what is seen in Hardware Inventory. If this directory is found, then it will be manually assigned to the product property `additional_drivers`. The directory name `byAudit` is case sensitive. The directory names for `Vendor` and `Model` are not case sensitive (Dell and dELL are treated the same way).

Some manufacturers use model names that are very delicate to this method, since some special characters such as `/` are not allowed to be used in files or directory names. An example for a model name could be: "5000/6000/7000". A directory with this name is not allowed because of the special characters. Since the third Service Release from opsi 4.0.3 the following special characters: < > ? " : | \ / * were replaced internally with an underscore "_" character. With this change can the above example be replaced with: "5000_6000_7000" the directory will automatically be shown, even though the directory structure information in the hardware inventory is not visually the same.

⚠️ After uploading a driver to `.drivers/drivers` or `.drivers/drivers/preferred` the script `create_driver_links.py` should be executed on the `opsi-depot-server`!

## 4.13. Software inventory

With this tab you get the last known software information for this client (only available if a single client is selected).
4.14 Logfiles: Logs from client and server

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

The level up to which the log lines are seen can be chosen by a slider (wheel mouse enabled), so that errors can be easily found.

It's also possible to search in the log file (to continue the search press F3 or Ctrl-L = last search repeated).
4.15. Product default properties

To change the default values of the products for one or more opsi-depots, there is a tab, called *Product default-properties*. This is only available if you select *Properties of depots* (which is the second button at the top right hand side).

In the main table, all products are listed with the product version as well as the package version.

If a product is selected, at the top of the right side (as is customary for the client product configuration) general information about the product packages is shown. Below is the list of all depots, that have installed the selected product. The table below with the property keys and values is also known from the client product configuration.
You can select a single depot or multiple depots to change the default values (which are also called the depot values) of the product. As the default, all available depots are preselected. With the usual shortcuts (Ctrl-a, Ctrl-Click or Shift-Click) multiple or all clients can be selected.

If the property value is shown grayed (see Figure 38, “opsi-configed: product default properties” - “gui_language”), the values for that property differ on the selected depots. On the right side of the depots are three buttons:

- (=+): Mark all depots that have identical values
  All depots, that have the same default values, are marked.
- (++): All depots are marked.
- (globe): set the package default values
  The original package default values of the products will be set for the selected depot(s).

4.16. Host parameters in client and server configuration

There are many configuration options for the opsi server and the opsi clients that may be set or changed via the tab Host parameters. Server defaults are set in the mode server configuration, client specific values in the mode client configuration plus manual selection of the Host parameters tab (see also Section 4.4, “opsi-configed modes Client configuration / server configuration / license management”).

On principle, these configuration entries (config objects of the opsi-server) are conceived as lists of values. Therefore they are edited via the list editor component (cf. Section 4.10, “Property tables with list editor windows”).

Depending on the specific type of a configuration object,
• the elements of the list can be of type text (Unicode) or of type Boolean (i.e. true/false);
• the set of all values from which elements can be selected may be fixed or extensible.
• the object has a defaultValues-entry, which comprises in the singleValue case exactly one list
element, in the multiValue case some partial list selected from the list of all possible values.

New configuration objects can be created via the context menu of the server host parameters. At this
place it is also possible to remove existing entries.

The relationship of server and client entries is a little bit complicated.

• Server entries hold the defaults for client entries (this is the defaultValues entry in the server
configuration object)
• Consequently, if a client entry is needed a server configuration object must be created
beforehand.
• And, when a server entry (a config object) is deleted, the depending client entries (called config
states) are (automatically) deleted as well.
• If the client related value is shown as identical with its (server based) default value this may be be
because there does not exist a client specific data base entry or because the client value is
identical with the server default value. In the first case the client value changes when the server
default value gets new contents, in the second case the client remains unchanged.
• In order to work comfortable with this situation since opsi-configed version 4.0.7.6.5 the context
menus of the client host properties offer the options (1) to remove the specific client values, so
that from now on only the current server values are decisive, (2) to fix the specific client values to
the current server values.
• if the currently presented client value is not identical with the server default value it is given in a
bold font.
• There are configurations objects for which client values can be created and edited but in fact only
the server objects are used. In most cases, the current configed does not show them any more in
the client parameters view.

To get more structure the configurations objects are categorized in some (predefined) groups. The
groups are listed in a tree-like manner on the left part of the panel. The entry name/value pairs
belonging to the selected group are shown in the right part of the panel. Wheel mouse scrolling is
enabled as well on the left as on the right side.
4.16.1. Management of user rights and roles

Starting with version 4.0.7.5 the *opsi-configed* includes the user roles function.

In order to use this feature the module *user roles* must be activated in the modules_-file.

In the interface, in the overview of the server host parameters, the category *user* shows the availability of the function (not necessarily active). The *user* branch of the properties tree starts with a boolean entry

```
user.{}.register
```

with default value *false*.

The other entries at this location represent the default values for the user-specific configurations of
the server console (cf. Section 4.20, “Server-Console”).

To activate the user role extension you need to:

1. Set the value of user.{}.register to true.
2. Load a modules file that has the userroles extension temporarily or permanently activated.

When the user-role extension is activated, an entry is created in the properties tree for the logged-in user. The default settings used for the administration of rights are like the “classic” requirements for an administrator, that means, that this user has no restriction whatsoever. E.g., for a user named admindepot1 the following entries are generated:

```plaintext
user.{admindepot1}.privilege.host.all.registered_readonly   [false]
user.{admindepot1}.privilege.host.depotaccess.configured     [false]
user.{admindepot1}.privilege.host.depotaccess.depots         []
user.{admindepot1}.privilege.host.opsiserver.write           [true]
```

These four items mean:

- `admindepot1` is not restricted to read-only access to the server (a pure read-only access might be appropriate for a help desk staff member);
- depot restrictions do not exist or are not taken into account;
- consequently, the list of depots available to the user can stay empty (and if some depots are entered, this has no effect);
- the user is allowed to edit config server settings of all kinds.

In the case that the access of `admindepot1` is to be restricted to the computers in the depot server `depot1`, the following should be set:

- `host.depotaccess.configured` is to be set to true;
- the value “depot1” is to be put into the list `host.depotaccess.depots`.

After a complete data reload, clients from other depots are not more visible to `admindepot1` (and also only the depot settings for `depot1` are accessible).

`admindepot1` him/herself can change this settings as long as she/he owns the privilege `host.opsiserver.write`

In order to complete the restriction, it therefore is required to set

- `host.opsiserver.write` to false.
The privileges which are set in this way restrict only the functionality of the opsi-configed. Until further notice, they have no effect if the JSON-RPC interface of the opsi-server is accessed by other means.

4.17. Depot configuration

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

see also:

![Figure 41. opsi-configed: Tab Depot configuration](image)

4.18. Group actions

The button "group actions" in the main button bar (cf. Section 4.4, “opsi-configed modes Client configuration / server configuration / license management”) opens a window for group related functions.

At the moment, it provides only one function which is relevant for the opsi-localimage module.

- to search for an operating system, that had been installed on all of the clients of the selected group and therefore can be offered for all of the clients of that group.
4. opsi-Management GUI: opsi-configed

4.19. Product actions

The button "product actions" in the main button bar (cf. Section 4.4, “opsi-configed modes Client configuration / server configuration / license management”) opens a window for functions related to products resp. packages.

Currently it offers two options:

- An .opsi file (opsi package) can be selected or entered and can be uploaded to the opsi server; the default upload directory on the server is the network (samba) share named opsi_workbench. The button click starts installing the package on the server, like invoking the opsi-package-manager.

- The WinPE files and install files for a Windows product (Windows Vista and above) can be uploaded to the server product directory (share opsi_depot), so that Windows products do not have to be managed from the server side.
4.20. Server-Console

Some of the following features require at least python-opsi version 4.0.7.38, in particular defining your own commands as described in Section 4.21, “Define commands” and using them via configed.

With version 4.0.7.5, the configed is extended with a new main menu entry, the "Server Console". At this place some options are bundled to access the opsi-server via a SSH-Connection. It is as well possible to start a terminal as well as menu items are offered of some predefined commands on the opsi-server.
4.20.1. Connection data and permissions

If not otherwise configured, it is tried to build a SSH connection with the same user/server pair for which to configed login was done.

Should this not be the case the connection can be also started via a SSH key (possibly with a password) when the configed starts. In this case, the following start parameters can be used:

- --ssh-key PATH: e.g. --ssh-key /home/user/.ssh/id_rsa
- --ssh-passphrase PASSPHRASE: e.g. --ssh-passphrase Password

The settings can be changed or adjust under the menu entry "Connection Information".

The visibility of menu items in the server console menu is controlled by a series of server host parameters in the user section. If the user roles feature is used (cf. Section 4.16.1, “Management of user rights and roles”) the configs are specifically set for each user (the default values for a newly created user entry are taken from the top user level).

In order to be able to use different functions, the appropriate server settings must be activated.

- user.{}.ssh.serverconfiguration.active:
  Activates the ssh connection settings menu. (Default: false)
- user.{}.ssh.commandmanagement.active:
  Activates the editing of commands and their menu entries. (Default: false)
- user.{}.ssh.menu_serverconsole.active:
  Deactivate the "Server Console" menu in principal. (Default: true)
- user.{}.ssh.terminal.active:
  Allows the usage of the ssh shell. (Default: true)
- user.{}.ssh.commands.active:
  Allows to execute all menu items displaying stored commands. (Default: true)
4.20.2. SSH-Terminal

With the Terminal, Linux commands can be run from the connected SSH-Server. In addition to the possibility to replace the input with asterisks (*), which is strongly recommended for the input of passwords, a process can also be canceled by clicking the "End process / connection" button or by pressing "Ctrl + C".

Just like in the Terminal, the "TAB" can be used to complete commands. Warning: Paths will not be completed - only Linux system commands.

Besides it is also possible to specify data sources, that before the execution can be replaced by concrete data. More about this functionality: Section 4.21, “Define commands” - Item: Datasources

![SSH-Terminal](image)

Figure 45. opsi-configed: SSH-Terminal

4.20.3. Predefined commands with input masks

Under the menu group "opsi" a few commands are available independently of the self-defined commands with their own input interface. These simplify the handling of various scripts.

- **Download from data ...**
  Any data file which can be downloaded from the Internet can also be downloaded by the "wget"-command and stored in a certain path on the Server. This could be used for example to download opsi-packages from download.uib.de

- **Create opsi product file ...**
  Prerequisite for this command is an opsi-utils package with version >= 4.0.7.7. Using this menu item, an opsi package can be created, specifying the directory upon which must be stored. In addition, can the found versions (package and product version) be displayed and overwritten with a button. And also, a md5sum and/or a zsync file can be created.

- **Set opsi-rights ...**
This menu item maps the opsi command opsi-set-rights. After entering a specific (optional) path in which the script is to be executed, the root password is prompted and the script is executed in a separate window.

• Package-Installation ...
With this command, opsi packages can be installed on all depots or in one depot using "opsi-package-manager". You can also specify the server path to the package where the opsi package is located.

By selecting a package from the Internet, the functionality of "File download ..." command is taken up and then the downloaded package is installed on the depot. Additionally, the parameters "--update" and "--setup" of the opsi-package-manager are implemented. If the zsync and md5 files of an opsi package are to be downloaded, the switch "zsync and md5 include" can be activated. Then the url of the packages is added accordingly and the additional files are also obtained.

You can find more about opsi-package-manager under Section 5.3.2, “Tool: opsi-package-manager - (de-)install opsi-packages”

• Package-Deinstallation ...
From a list of installed packages one can be selected and uninstalled.
Please check Section 5.3.2, “Tool: opsi-package-manager - (de-)install opsi-packages”

• Deployment opsi-client-agent ...
If you want to add existing computers to opsi, the opsi-client-agent must be installed on the target computer. If you select the clients in the configed and execute this command, the client names are copied into the corresponding field. If the command is to be executed on several clients in a single call, the login data must be the same on all the participating computers.

Attention: The location of the script have to be: "/var/lib/opsi/depot/opsi-client-agent/opsi-deploy-client-agent" Detailed information can be found in the opsi-getting-started manual on the First steps chapter.

TIP: Some user interfaces include a selection component for paths in the directory structure. If the button "Find Subdirectories" is activated, all directories or files that are contained in the specified path will be listed. To visualize further sections, you can press the button several times. This functionality is, among others, in the "Set Opsi rights" or the "Package installation" interface.

4.21. Define commands

In addition to the predefined server console commands, you can create or remove your own commands, which can be accessed via menu items. It should be noted that different Linux systems may not be able to execute the same commands. Thus, the administrator must be sure that the commands can be executed on the addressed Linux system.
Figure 46. opsi-configed: Define commands

Following data must be or rather could be (marked with a "*") for a command:

- **Menu-Text:**
  When creating a new command you must make sure that the menu text has not been used already for another command. If a menu text is to be changed, the command must be first deleted with the minus button, and then the new command can be entered.

- **Description***:
  If a more detailed description is stored, then it appears as a tool tip text on the command.

- **Superior menu***:
  Determines in which menu the new command should appear as a menu item. In the case that field is empty, then the menu entry will be directly assigned to the "Server console" menu.

- **Position***:
  The position determines the order (small numbers comes first) of the menu points in total, and thus within each respective menu. If alphabetical order should be displayed, all items must be set identically (e.g., all 0). Should the the field remain empty, then the position 0 is assigned by default.

- **"Sudo" rights***:
  If one of the commands in the command list requires administrative rights, a check mark must be set on "Required root privileges" afterwards the commands in the list are automatically executed with the keyword "sudo".

- **Command list:**
For the command list, the Linux commands must be entered line by line, so that they can be executed sequentially. Caution: Command can be tested or executed on the SSH server by means of a button without creating an extra menu point.

• Data sources* (on the command list):
  Additionally methods can be stored as a data source. Before the command can be executed, the parameters are overwritten with the result of the applied method. The following parameters are possible:

  ◦ Interactive input:
    It is possible to specify parameters for the commands or to identify them for an interactive output. This is done with the following format "<<< This text will be displayed to the user and replaced by the user input >>>", it is recommended though to write a sample input for the parameter for the user text.

  ◦ Selected client names / Selected client IP addresses

  ◦ Selected depot names / Selected depot IP addresses

  ◦ configserver name

  ◦ Connected SSH servername

  Note: Except for "Interactive Input", the return of the methods can also be formatted, for example, into a comma separated list. In the interface, the data source can be tested, and also insert it into the location marked in the field of the command list.

![Figure 47. opsi-configed: Execute command - Parameter query](image)

- On Linux, commands can be combined using two commercial ANDs ("&&"). However, it must be ensured that the second command, if needed, is executed with administrative rights, since this is not done automatically. Example: Requires root privileges: "activated", command list: "apt-get update --yes && sudo apt-get upgrade --yes". +

- During the execution, no user input can be made. It is necessary to control all the inputs via the command \ parameters (example: "--yes" option for "apt-get upgrade")
5. opsi-server

5.1. Overview

The functionality of an opsi-server can be installed on many standard Linux distributions.

There are two different major roles, which can be combined on one server:

- **opsi-config-server**
  The functionality of the configserver includes the storage and processing of the configuration data in different backends and provisioning this via a web service and on the command line.

- **opsi-depot-server**
  The functionality of the depotserver includes storing the actual installation files of the software to be distributed, operating systems, boot images and making them available to the client via smb/cifs, https, tftp.

The hardware requirements for these services are generally low, so operating an opsi-server in a virtualization environment is not a problem.

5.1.1. Installation and initial operation

The installation and starting of a opsi-server is explained in detail in the separate opsi-getting-started manual.

5.1.2. Samba Configuration

In order to give the client computers access to the software packages, the opsi-server provides shares that can be mounted by the clients as network drives. For Windows Clients the shares are provided by Samba. To configure your samba according to the needs of opsi (or to repair) call:

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

After changing the Samba configuration it is necessary to restart the Samba service (systemctl restart smbd.service).

5.1.3. The opsiconfd daemon

Opsiconfd is the central configuration daemon of opsi. All client components (opsi-client-agent, opsi-configed, opsi-linux-bootimage, ...) connect to this service to access the configurations in the backends. The opsiconfd is configured via the file `/etc/opsi/opsiconfd.conf`. The individual configuration options are documented in this file. Here are some additional notes.

- `[global] admin networks +:`
  With this option the administrative access to the opsiconfd can be restricted to connections from
certain network addresses. Multiple network addresses can be specified separated by commas. Non-administrative client connections can also be made from other networks.

- **+[global] max log size +:**
  This option can be used to limit the size of the log files. For historical reasons, this is limited to 5MB per log file. Since opsi 4.0.6 it is possible to change this value to your own needs. To disable the size limit, the value can be set to 0.

In addition to this setting option, it is possible to use the logrotate tool to automatically compress and rotate the log files on a server. Please refer to the associated manual for configuration options.

If the maximum size of the log files is known, you can calculate how much storage space the logs will need. There are five different, client-related log types that are written by opsiconfd: `bootimage`, `clientconnect`, `instlog`, `opsiconfd` and `userlogin`. There are also some client-independent logs: `opsiconfd.log`, `opsipxeconfd.log`, `opsi-backup.log`, `opsi-package-updater.log` and `package.log`.

If we assume a configuration implemented with opsiconfd and logrotate, in which all log files are limited to 5MB, apart from `package.log`, which can be 10MB, then we end up with the following calculation:

\[
\text{(number of clients} \times 5 \times 5\text{MB}) + 5\text{MB} + 5\text{MB} + 5\text{MB} + 5\text{MB} + 10\text{MB}
\]

For 100 clients we should thus reserve 2530MB for the opsi logfiles. Because Logrotate is usually only active at a certain time, we recommend rounding up this number.

### 5.1.4. Required system user accounts and groups

- **User opsiconfd**
  This is the user under which the opsiconfd daemon runs.

- **User pcpatch**
  This is the user the `opsi-client-agent` uses to mount and read from the `depotshare`. By default this user has the home directory `/var/lib/opsi`. You may set the password for this user by `opsi-admin -d task setPcpatchPassword`.

- **Group pcpatch**
  In addition to the user pcpatch, there is also the group pcpatch. Most files are fully accessible for both the user and the group. The system administrators of the opsi-server should therefore be members of the pcpatch group.

- **Group opsiadmin**
  The members of this group can authenticate themselves to the opsi-webservice and thus e.g. work with opsi-configed. Therefore, all opsi administrators should be a member of this group.
5.1.5. Required shares

- **Depotshare** with software packages (*opsi_depot*)
  The software packages prepared for installation by the opsi-winst program are located on the depot share.
  By default, this directory is located on the opsi server as `/var/lib/opsi/depot`.
  Below this directory there is a directory for each software package with the name of the software package.
  The installation scripts and files are then below these directories.

  This directory is shared read-only as *opsi_depot*.

  - In older versions of opsi the corresponding directory was `/opt/pcbin` and the share was called *opt_pcbin*.

- **Working directory to build packages** (*opsi_workbench*)
  At `/var/lib/opsi/workbench` you will find the area to create new packages and from where you should install packages with the opsi-package-manager. This directory is shared as *opsi_workbench*.

  - Since opsi 4.1 the path can be configured per depot through the attribute `workbenchLocalUrl`.

- **Configuration files of the file-backend** (*opsi_config*)
  The configuration files of the file backend are located under `/var/lib/opsi/config`. This directory is shared as *opsi_config*.

  - If you work on the files via this share, do not use editors that change the file encoding (Unix/DOS), and remove backup files such as *.bak.*

5.1.6. opsi PAM Authentication

opsi uses various PAM-modules for user authentication. So far, different PAM modules have been used for different distributions. The PAM modules used are listed in the following list:

Default: `common-auth`
openSUSE / SLES: `sshd`
CentOS and RedHat: `system-auth`
RedHat 6: `password-auth`

As you can see from the list, various PAM configurations were used, but these can change again depending on the local PAM configuration. Since these adjustments always required an intervention in the sourcecode, you can now create the *opsi-auth* file under: `/etc/pam.d/` and save your own PAM configuration for opsi. If this file exists, opsi automatically uses this configuration.

The following simple example should clarify the behavior: If you run a Debian/Ubuntu system and
receive a PAM error message when logging on to opsiconfiged, although an SSH connection to the server can be opened with the same user data, you can create the file /etc/pam.d/opsi-auth with the following content:

```bash
@include sshd
```

After restarting opsiconfd, opsi will automatically use the sshd-PAM module for authentication.

Please note that the application of the ACL uses case-sensitive interfaces, whereas authentication via PAM can be case-insensitive. This can result in the fact that despite successful authentication, it is not possible to work with the service because the ACL prevents this.

### 5.2. Notes on switching to Samba 4

When Samba 4 had reached stable, the development and maintenance work for the Samba 3 branch was discontinued. As a result, almost all common Linux distributions (client and server versions) come with Samba 4 instead of Samba 3. This results in some changes that will be documented in this chapter.

Samba shares are an essential component for the functionality of opsi. Due to the “general” update to Samba 4, there are a few things to consider that will be briefly explained in the following chapters.

A distinction must be made in which operating mode Samba is executed. A special feature of Samba 4 is the ability to run as a fully-fledged Active Directory-compatible domain controller. In this operating mode (which is called PDC mode in the following chapters for reasons of simplification), there are restrictions that had to be adopted from Active Directory for reasons of compatibility. Most distributions are usually equipped with Samba 4, but only in the normal shares operating mode. Operating a full-fledged Active Directory domain is generally not possible with the standard packages from the distributions. An exception here is the Univention Corporate Server, in which the PDC mode is also integrated in the standard packages.

#### 5.2.1. /etc/opsi/opsi.conf: pcpatch and opsifileadmins

The restrictions described in this chapter only affect the PDC mode of Samba 4.

The classic installation variant with the user: pcpatch with the primary group: pcpatch cannot be followed for installations with Samba 4. Since Samba 4 is subject to the basic restrictions of Active Directory, groups with the same name as users (as is usual in Unix/Linux) are no longer allowed. For this reason, a new configuration file has been introduced for Samba 4 installations: /etc/opsi/opsi.conf, which controls how the group is determined for Samba access to the shares. In the case of Samba 4 installations, the group name pcpatch is now renamed via this file and is now called opsifileadmins. This means that the users which must have access rights to the shares of opsi (opsi-packagers) under Samba 4 cannot become a member of the pcpatch group, but must be a
Furthermore, in this case the user **pcpatch** must now be created as a fully-fledged domain user and no longer as a system user, since otherwise he cannot access the domain shares.

These steps are carried out automatically when installing opsi on a Univention Corporate Server, if the installation process detects that Samba 4 is running in PDC mode.

Besides the UCS installations, currently there are no other default Active Directory configurations. So these steps have to be done manually for any other Samba 4 Active Directory domain controller installation. During future updates, the opsi system checks for the required user configuration and does not try to create users, that already exist.

For any questions please contact opsi support. In case you do not have an opsi support contract, please contact info@uib.de.

### 5.2.2. Shares Configuration

The changes described in this chapter are relevant for all operating modes of Samba 4.

In Samba 3 the default setting was, that every file or directory was executable by the clients. This behavior has been completely changed in Samba 4. Now all files, that should be executable from the share, must also have the executable bit set on the Unix side.

This is a problem for the operation of opsi. It is not possible to circumvent this behavior via the opsi rights management, as this would require a complete revision of the opsi rights system. This is not possible in opsi 4.

There are two ways to work around the problem with opsi 4.0:

**Option 1 (recommended):** You can set the following option in smb.conf:

```plaintext
acl allow execute always = true
```

This option restores the behavior of Samba 3 for the relevant shares.

This option can be set for individual shares as well as globally. We recommend that you do not set this globally but individually for all opsi shares (unless this has already been set automatically).

This variant may not work with Univention Corporate Server, since a highly modified Samba 4 variant is used here. In this case, use option two.

**Option 2:** this behavior can be overridden on the affected shares via the individual share configuration using the following option for each member of the pcpatch group (share users):

```plaintext
```
admin users = @pcpatch

Opsi has been using this fix for a while with UCS >= 3 and Samba 4. With this fix, the Samba process of the user is executed with elevated rights.

Opsi automatically sets this option for the \textit{opsi_depot} share for Samba 4 distributions via \texttt{opsi-setup --auto-configure-samba}. Since this share is only mounted read-only, the security risk is relatively low.

For all other shares, that are mounted as read/write, it should be borne in mind that with this fix the Samba process runs with elevated rights. This can be a potential risk. No exploits are currently known for this vulnerability, but of course this is no guarantee that such an exploit does not exist.

The Linux smb daemon has a bug. This is in combination with the \textit{opsi_depot} share definition in \texttt{smb.conf}. The oplock parameters must be removed on existing installations. New opsi installations and, accordingly, new shares are created without oplocks.

\textbf{5.2.3. Access to shares: clientconfig.depot.user}

This restriction affects all operating modes of Samba 4.

When using Samba 4, it may be necessary to explicitly specify the domain/user combination with which to mount the depot share. There is also the new setting: \texttt{clientconfig.depot.user}. If there is no such setting, the user \texttt{pcpatch} is used. The value of this setting has the syntax: \texttt{<domain name>\<user name>}

For example, the setting: \texttt{clientconfig.depot.user = opsiserver\pcpatch} indicates that when mounting the depot share the authentication will be done as domain \texttt{opsiserver} and as user \texttt{pcpatch}.

Such a setting can be created via opsi-configed: Server configuration / clientconfig / right mouse button: Add standard configuration entry.

Such a setting can also be created on the command line (whereby \texttt{pcpatch} must be replaced by the desired value e.g. \texttt{opsiserver\pcpatch}):

\begin{verbatim}
opsi-admin -d method config_createUnicode clientconfig.depot.user "clientconfig.depot.user" pcpatch
\end{verbatim}

This system-wide setting can be customized per client (e.g. in configed in the Host parameters tab).

\textbf{5.3. opsi command line tools and processes}
5.3.1. Tool: \textit{opsi-setup}

The \textit{opsi-setup} program is the "Swiss army knife" for setting up opsi. 

For example, the packages for installing opsi on the server use this script to set up opsi correctly. Since this script can also be called up externally, various maintenance work and corrections can be carried out:

- Registration of an opsi server as a depot server
- Correct file access rights
- Initialize backends
- Upgrade backends (from 3.4 to 4.0)
- Configure the \textit{MySQL backend}
- Set the default configuration
- Clean up backends
- Set the Samba configuration
- Set the DHCP configuration

The command \texttt{opsi-setup --help} shows the program options:

```
# opsi-setup --help

Usage: opsi-setup [options]

Options:
   -h, --help  show this help
   -l          log-level 0..9
   --log-file <path>             path to log file
   --backend-config <json hash>  overwrite backend config hash values
   --ip-address <ip>             force to this ip address (do not lookup by name)
   --register-depot              register depot at config server
   --set-rights [path]           set default rights on opsi files (in [path] only)
   --init-current-config         init current backend configuration
   --update-from=<version>       update from opsi version <version>
   --update-mysql                update mysql backend
   --update-file                 update file backend
   --configure-mysql             configure mysql backend
   --edit-config-defaults        edit global config defaults
   --cleanup-backend             cleanup backend
   --auto-configure-samba        patch smb.conf
   --auto-configure-dhcpd        patch dhcpd.conf
   --renew-opsiconfd-cert        renew opsiconfd-cert
   --patch-sudoers-file          patching sudoers file for tasks in opsiadmin context.
```
The functions and parameters in detail:

- **--ip-address <ip>**
  This option is used to specify an IP address for the `opsi-server` and to bypass the name resolution.

- **--register-depot**
  This option is used to register an `opsi-server` at another `opsi-server (opsi-config-server)` as a depot server. For details, see Section 9.15.2, “Creating an depot server”.

- **--set-rights [path]**
  Sets or corrects the file access rights in the main opsi directories:
  - `/tftpboot/linux`
  - `/var/log/opsi`
  - `/var/lib/opsi`
  - `/var/lib/opsi/depot`
  - `/var/lib/opsi/workbench` (or a differently configured workbench location on the depot)
  - `/etc/opsi`

  A directory can also be specified as a parameter. Then the rights of all opsi-relevant directories and files below this directory are set, e.g.:
  ```bash
  opsi-setup --set-rights /var/lib/opsi/depot/winxppro/drivers
  ```

- **--init-current-config**
  Initializes the backend configuration. This should be executed after every change to `/etc/opsi/backendManager/dispatch.conf`.

- **The commands:**
  - **--update-mysql**
  - **--update-file**
    are used to upgrade the respective backends (e.g. from opsi 3.4 to opsi 4).
    See the `releasenotes-upgrade-manual` for details.

- **--configure-mysql**
  Used to initialize the `{MySQL backend}` for the first time.
  For details, see Section 5.6.2.3, “Initializing the MySQL-Backend”.

- **--edit-config-defaults**
  To edit the default values of the configuration as shown in server configuration in `opsi-configed`.
Figure 48. Dialog: `opsi-setup --edit-config-defaults`

e.g.:

**clientconfig.depot.id**

The name of the default depot server.

**clientconfig.depot.drive**

The drive letter used for mounting the share with the installation data. You can either select a drive letter or **dynamic**. With **dynamic** the client will try to automatically select an drive letter from those not in use.

**license-management.use**

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

**product_sort_algorithm**

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

- **--cleanup-backend**
  Checks the current backends for integrity and removes obsolete or unreferenced entries. Examples for entries that may be removed are products without reference (not installed on depot / client), host-groups without a parent and configstates without corresponding config.

  It is common good practice to create a backup through `opsi-backup` before cleaning the backend.

- **--auto-configure-samba**
  Creates the opsi share entries in the `/etc/samba/smb.conf` configuration file

- **--auto-configure-dhcpd**
  Creates the by opsi needed entries in the configuration file of the DHCP daemon.
Don't use this if you not plan to use the dhcpd on the opsi server.
More details in the opsi-getting-started manual

5.3.2. Tool: opsi-package-manager - (de-)install opsi-packages

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

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

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

Install a package (asking no questions):

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

Install a package (asking questions):

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

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

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

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

```bash
opsi-package-manager -s -i softprod_1.0-5.opsi
```

Or:

```bash
opsi-package-manager --setup-with-dependencies --install softprod_1.0-5.opsi
```

Uninstall a package (asking no questions):

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

Extract and rename a package:

```bash
opsi-package-manager -x opsi-template_<version>.opsi --new-product-id myprod
```
It is possible to install a package with a different product id. This is helpful if a custom Windows netboot product was derived from an existing package and that package was updated in the meantime.

```bash
opsi-package-manager --install win7-x64_1.2.3.opsi --new-product-id win7-x64-custom
```

Please note that products installed that way will not be automatically updated by opsi-package-updater.

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

Please note:

- The option -d or --depots are reserved for the use in a multi-depot-server environment.
- Using option -d the opsi-package will be copied to the /var/lib/opsi/repository directory of the target server before installing. Please make sure that there is enough free space on this file system.
- If installing new packages fails because there is not enough free space available in your temporary folder you can specify a different temporary folder with the --temp-dir option.

see also:

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

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

Manage opsi packages

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

Options:
- v, --verbose increase verbosity (can be used multiple times)
- q, --quiet do not display any messages
--log-file <log-file> path to debug log file
--log-file-level <log-file-level> log file level (default 4)
- d, --depots <depots> comma separated list of depot ids to process
```
5.3.3. Tool: **opsi-package-updater**

The command line utility **opsi-package-updater** is designed to download and install comfortable opsi packages from one or more repositories or another opsi server. Using opsi-package-updater makes it easy to keep the products on an opsi server up to date. It may be also used in a cronjob to keep depot server in sync with the primary server.

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

Each repository can be configured in terms of access information and behaviour.

General configuration will be done at the configuration file `/etc/opsi/opsi-package-updater.conf`.
Usage

--help shows a small help. opsi-package-updater works with different modes. Each brings it's own help with them.

```bash
# opsi-package-updater --help
usage: opsi-package-updater [-h] [--version] [--config CONFIGFILE]
               [--verbose | --log-level {0,1,2,3,4,5,6,7,8,9}]
               [--force-checksum-calculation]
               [--repo repository_name]
               [--use-inactive-repository]
               {install,update,download,list} ...
```

Updater for local opsi products.

optional arguments:
- `-h, --help` show this help message and exit
- `--version, -V` show program's version number and exit
- `--config CONFIGFILE, -c CONFIGFILE` Location of config file
- `--verbose, -v` Increase verbosity on console (can be used multiple times)
- `--log-level {0,1,2,3,4,5,6,7,8,9}, -l {0,1,2,3,4,5,6,7,8,9}` Set the desired loglevel for the console.
- `--force-checksum-calculation` Force calculation of a checksum (MD5) for every package. Default is to use existing checksums from the .md5-file of a package if possible.
- `--repo repository_name` Limit the actions the given repository.
- `--use-inactive-repository` Force the activation of an otherwise disabled repository. The repository must be given through --repo.

Mode:
{install,update,download,list}
- `install` Install all (or a given list of) downloadable packages from configured repositories (ignores excludes)
- `update` Update already installed packages from repositories.
- `download` Download packages from repositories. This will not install packages.
- `list` Listing information

Modes have their own options that can be viewed with MODE -h.
There are some common options.

- **--verbose** increases the amount of shown messages. This can be given multiple times to increase verbosity. A specific log level can be given through **--log-level**. Both options only affect the output in the terminal.

- **--repo <name of a repository>** limits the actions to the given repository. Available repositories can be shown with **list --active-repos**.
The different modes result in different behaviour. The modes `install`, `update` and `download` load packages from a repository, whereas `list` is used to show information.

The mode `install` installs new packages. The mode `update` updates already installed packages. Both modes require no further parameters.

Example: Installing all available packages from all repositories:

```
opsi-package-updater install
```

The modes `install` and `update` allow to limit the actions to specific products by handing over their ID.

Example: Updating the packages for the products `firefox` and `javavm`:

```
opsi-package-updater -vv update firefox javavm
```

In combination with the `--repo` switch the package source can be limited.

Example: Installing the package for `ubuntu` from the repository `uib_linux`:

```
opsi-package-updater -vv --repo uib_linux install ubuntu
```

The mode `download` allows to download packages without installing them afterwards. The switch `--force` forces the download of a package even though this version is already installed on the server.

Through `list --active-repos` the active repositories are shown. The information consists of name, address and if applicable the description of the repository.

Through `list --products` the available products per repository are shown.

To show the possible updates use `list --updatable-products`. This only considers already installed products. After that the update can be started through `update`.

```
opsi-package-updater list --updatable-packages
opsi-package-updater -v update
```

**Repository Configuration: Access**

Repository configurations are to be found in `/etc/opsi/package-updater.repos.d/`. You will find a commented template with all possible configuration options in there as `example.repo.template`.

There are two kinds of repositories - `internet repositories` and `opsi-server`.

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

- **baseURL** (z.B. http://download.uib.de)
- **dirs** (a list of directories e.g. opsi4.0/produkte/essential)
- and for password protected repositories **username** and **password**
- as an alternative to password protected repositories, you can use certificate authentication if the repository has this possibility. For that you need to configure **authcertfile** and **authkeyfile** with the full path to the client-certificate file and its keyfile.

Access through a proxy can be configured if required. To use a common proxy for all repositories configure it in **opsi-package-updater.conf**. This requires at least opsi-utils 4.1.1.33. All repositories without a separate proxy configuration will make use of it.

```plaintext
baseUrl = http://download.uib.de
dirs = opsi4.0/products/localboot
username =
password =
proxy =
```

**opsi-server**

A repository is of kind **opsi-server** if the repository configuration has the ID of another opsi-server set at **opsiDepotId**.

```plaintext
opsiDepotId = mainserver.my.lan
```

On a **opsi-depot-server** there is usually the central **config server** set. This results in **opsi-package-updater** fetching the packages from the directory /var/lib/opsi/repository of the given server.

**Repository Configuration: Behaviour**

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

- **autoupdate**: Newer versions of installed packages will be downloaded and installed
- **autoinstall**: Also packages which are not installed yet, will be downloaded and installed
- **autoinstall**: For all new installed packages and all clients on which these packages are installed the action request will be set to setup.
- **onlyDownload**: New packages are only downloaded and no further actions are done. A common use case is to use this option together with notifications so that after the download of new packages a mail informs an administrator about the new packages and the administrator will manually install the new packages at a later time.
In addition it is possible to send all these clients a Wake-On-LAN signal to install the new software to the clients. Using the opsi-product *shutdownwanted* you can make sure that the clients will be powered off after the installation.

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

5.3.4. Tools for opsi API access: opsi-admin & opsi interface page

opsi 3.0 introduced a python library which provides an API for opsi configuration. The *opsiconfd* provides this API as a web service that can be accessed in multiple ways.

**In the browser: opsi interface page**

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

![opsi config interface: Access to the web service via browser](image)

**On the commandline: opsi-admin**

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

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

Usage: opsi-admin [options] [command] [args...]

Options:
- h, --help           Display this text
- V, --version        Show version and exit
- u, --username       Username (default: current user)
- p, --password       Password (default: prompt for password)
- a, --address        URL of opsiconfd (default: https://localhost:4447/rpc)
--opsirc         Path to the opsirc file to use (default: ~/.opsi.org/opsirc)
An opsirc file contains login credentials to the web API.
-d, --direct         Do not use opsiconfd
- d, --no-depot       Do not use depotserver backend
- l, --log-level      Set log level (default: 3)
0=nothing, 1=essential, 2=critical, 3=error, 4=warning
5=notice, 6=info, 7=debug, 8=debug2, 9=confidential
-f, --log-file       Path to log file
- i, --interactive    Start in interactive mode
- c, --colorize       Colorize output
- s, --shell-output   Simple output (only for scalars, lists)
-r, --raw-output      Raw output
--exit-zero          Always exit with exit code 0.

`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 a username and password. The name of the currently used user is used by default. A different username can be supplied with --username.

For the use in scripts you don't want to passing the password as this might be seen from different users via the process list. In such cases it is a good idea to make use of an opsirc file. Another alternative can be direct data access through the -d option.

`opsi-admin` has an interactive mode that can be started with -i. This will usually be used with -i for colorised display and maybe with -d for direct data access - the command to use is `opsi-admin -i -c -d` or short `opsi-admin -idc`. In interactive mode you have input support with the TAB key. After some input the TAB key 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.

Usage of a connectionsettings file - opsirc

Since version 4.1.1.30 `opsi-admin` offers the possibility to save the configuration for a connection to the webservice in a file. This allows using the webservice without the need to hand over the credentials on the commandline.
By default the search is expected to be at ~/.opsi.org/opsirc. With the option --opsirc the path to the used file can be changed. This allows preparing multiple differentially configured files.

An opsirc file has the following contents:

```
address = https://seeing.the.ramp:4447/rpc
username = tony
password file = ~/.opsi.org/tonys_secret
```

Everything in an opsirc file is optional. If the file is empty or does not exist default values will be used.

In the previous example the password is saved in the file ~/.opsi.org/tonys_secret and will be read from there. This file contains only the password.

It is possible to state the password inside the opsirc file aswell:

```
address = https://seeing.the.ramp:4447/rpc
username = tony
password = first900
```

**Typical use cases**

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

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

List of all clients

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

Client delete

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

E.g.:

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

Create client

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

E.g.:
opsi-admin -d method host_createOpsiClient "pxevm.uib.local"

Set action request

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

e.g.:

opsi-admin -d method setProductActionRequest win7 pxevm.uib.local setup

Attach client description

opsi-admin -d method setHostDescription "dpvm02.uib.local" "virtual client"

Listing the IDs of all clients

This uses the option -S so that every client is on its own line. Through filtering for the type OpsiClient we avoid displaying the IDs of servers.

The output can easily be used in other programs or calls.

opsi-admin -dS method host_getIdents '{}' '{"type": "OpsiClient"}'

Listing the products installed on clients

opsi-admin -d method productOnClient_getObjects '[["productVersion", "packageVersion", "installationStatus"]]' '{"installationStatus": "installed"}'

set pcpatch password

opsi-admin -d task setPcpatchPassword

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

5.3.5. Server processes: opsiconfd and opsipxeconfd

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

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

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

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

**opsiconfd monitoring: opsiconfd info**

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

![opsiconfd info](image)

*Figure 50. opsiconfd info: opsiconfd values from the last hour*

*Figure 51. opsiconfd info: opsiconfd values from the last day*

### 5.3.6. Server process: opsi-tftpd-hpa

The opsi-tftpd-hpa bases on the standard tftpd-hpa with the additional feature to handle *named pipes*.

By default the opsi-tftpd-hpa is configured to run with an own systemd service file.

As the packages is sinstalled by the installer, the package features a simple level of verbosity. To change this level and get additional information one has to do the following:

```
# systemctl edit --full opsi-tftpd-hpa.service
```

The parameter *-v* has to be replaced by *--verbosity 7*. Afterwards the service needs a restart.

```
# service opsi-tftpd-hpa restart
```

*⚠️ on Debian 8 the edit operation is not available. However there is a workaround:*
# cp /lib/systemd/system/opsi-tftpd-hpa.service /etc/systemd/system/opsi-tftpd-hpa.service
# vi /etc/systemd/system/opsi-tftpd-hpa.service
# systemctl daemon-reload
# service opsi-tftpd-hpa restart

5.4. Web service / API methods

5.4.1. object oriented methods

Since opsi 4 we have to different kinds of API methods:

- **object oriented** methods
- **action oriented** methods

Overview

The opsi4 backends are based on objects. An object has a set of attributes.

As an example serves here the object *Product*. The object of type *Product* which describes the opsi package *javavm* looks for example like this:

```
"ident": "javavm;1.6.0.20;2"
"id": "javavm"
"description": "Java 1.6"
"changelog": ""
"advice": ""
"userLoginScript": ""
"name": "SunJavaRuntimeEnvironment"
"priority": 0
"packageVersion": "2"
"productName": "javavm"
"productVersion": "1.6.0.20"
"windowsSoftwareIds": null
"productClassIds": null
"type": "LocalbootProduct"
"licenseRequired": false
"setupScript": "javavm.ins"
"updateScript": ""
"uninstallScript": "deljvm.ins"
"alwaysScript": ""
"onceScript": ""
"customScript": ""
```

For each object there is a set of operations. Usually these are:
• **getObjects**: returns a list of objects that match the specified filter. If a list of attributes is specified, only these will be populated from the backend.

• **getHashes**: Like **getObjects**, except that unchecked raw data from the backend is delivered directly. Works more performant than **getObjects**, but should be used with caution.

• **getIdents**: Returns a list of object IDs that match the specified filter. The **returnType** can be used to select the data structure of the elements in the result. Possible values are: **unicode**, **list** and **hash**.

• **insertObject**: Creates a new object. If the object already exists, it will be completely overwritten with the new values. Attributes that are not passed (or passed with the value **null**) will be set to **null** in the backend.

• **updateObject**: Updates an object. Attributes that are not passed (or passed with the value **null**), will **not** be changed in the backend. If the object does not exist, no change takes place, **no** object is created.

• **createObjects**: An object or a list of objects can be passed. Each object is passed internally to **insertObject**.

• **updateObjects**: An object or a list of objects can be passed. Each object is passed internally to **insertObject** if it does not exist yet, otherwise to **updateObject**.

• **create**: Creates a new object and takes all possible attributes as single parameters. Internally **createObjects** is used. Existing objects will be overwritten completely.

• **deleteObjects**: Deletes a list of objects. It is mandatory to pass a list. Only the attributes identifying the object (**type/id/ident**) are used to select the objects to be deleted.

• **delete**: Deletes the object identified by the specified parameters.

To update an object, it is usually a good idea to use the **updateObjects** method. For example, **productOnClient_updateObjects** can be used to update the state of a package on a client. Whether the object already exists or not is then irrelevant.

The names of the methods are composed of:

\[<object-type>_<operation>\]

Thus they differ from the **legacy** methods from opsi 3.x which usually start with **get**, **set** or **create**.

The **getObjects** methods have two optional parameters:

• **attributes**

• **filter**

The **attributes** parameter is used to query only certain attributes of the object, which can provide speed advantages. If a list of attributes is specified, only these are read from the backend. The remaining attributes are returned with the value **null**. Attributes that identify the object (**type/id/ident**) are always filled. If all attributes should be read (default), **null** or [] can be passed as **attributes**.

Example for the method **product_getObjects**, parameterized with **attributes:["name"]** for the package
(Product) javavm:

```
"onceScript": null,
"ident": "javavm;1.6.0.20;2",
"windowsSoftwareIds": null,
"description": null,
"setupScript": null,
"changelog": null,
"customScript": null,
"advice": null,
"uninstallScript": null,
"userLoginScript": null,
"name": "Sun Java Runtime Environment",
"priority": null,
"packageVersion": "2",
"productVersion": "1.6.0.20",
"updateScript": null,
"productClassIds": null,
"alwaysScript": null,
"type": "LocalbootProduct",
"id": "javavm",
"licenseRequired": null
```

If you don’t want to ask for attributes but instead you need to use the second parameter filter you have to pass the attribute parameter as 

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

The `filter` parameter can be used to filter the list of objects. For example, for `product_getObjects` the filter `{"id": "javavm"}` restricts the return to the object with the ID `javavm`. Multiple passed filter attributes are AND-linked here. If a list of values is passed for an attribute, this results in an OR operation. For strings, a * can be used as a placeholder. If you do not want to filter (default), you can pass `null` or `{}` as filter.

Example for `product_getIds` with filter `{"id": ["opsi-client-agent", "opsi-script"], "productVersion": "4.1*"} (returnType = hash):
For the methods to which one or more objects are passed, this must be done as a JSON object or a list of JSON objects.

The most important objects are:

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

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

This design:

- is created to transfer information about clients (several) faster
- filter data with a unified syntax
- allows to check all input for correct syntax

With this we got an increased stability and higher performance.

**host** *(server and clients)*

Example for a OpsiClient:

```json
method host_getObjects [] {"id":"xpclient.vmnat.local"}
[
    {
        "ident" : "xpclient.vmnat.local",
        "description" : "",
        "created" : "2012-03-22 12:13:52",
        "inventoryNumber" : "",
        "ipAddress" : "172.16.166.101",
        "oneTimePassword" : "",
        "lastSeen" : "2012-03-30 16:20:04",
        "hardwareAddress" : "00:0c:29:35:70:a7",
        "opsiHostKey" : "1234567890abcdef1234567890abcdef",
        "type" : "OpsiClient",
        "id" : "xpclient.vmnat.local"
    }
]
```

Most of this data is displayed on the clients tab of the opsi-configed.

Possible types are:
- **OpsiClient**
- **OpsiConfigserver** (which means implicit this is also a **OpsiDepotserver**)
- **OpsiDepotserver**

The server type have different and additional data.

Example for a server:

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

Most of this data is displayed on the **depot configuration** of the opsi-configed.

**group (group administration)**

Describes groups and their hierarchical structure.

Example for a group object:
method group_getObjects
[
  {
    "ident" : "sub2",
    "description" : "sub2",
    "notes" : "",
    "parentGroupId" : null,
    "type" : "HostGroup",
    "id" : "sub2"
  },
  {
    "ident" : "subsub",
    "description" : "subsub",
    "notes" : "",
    "parentGroupId" : "sub2",
    "type" : "HostGroup",
    "id" : "subsub"
  }
]

objectToGroup (group administration)

Describes the membership of an object in a group.

There are Hostgroups and Productgroups

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

**product (product meta data)**

Describes the meta data of a product which are defined while creating the package.

Example for a product object:
If you have multiple depot servers, you may have different versions of one product.

The entries `productClassIds` and `windowsSoftwareIds` are not used right now.

**productProperty (definition of product properties)**

Describes the properties of a product which are defined while creating the package.

Example for a `productProperty` object:
The real default values are stored in the context of the depot in a productPropertyState object.

**productPropertyState** *(depot or client specific product property settings)*

Describes:

- the default value of a product property on a given depot properties of a product which are defined while creating the package.
- the client specific settings of product properties.

Example for a productPropertyState objects:
method productPropertyState_getObjects [] {"productId":"jedit"}
[
  {
    "ident" : "jedit;start_server;sepiolina.vmnat.local",
    "objectId" : "sepiolina.vmnat.local",
    "values" : [
      false
    ],
    "type" : "ProductPropertyState",
    "propertyId" : "start_server",
    "productId" : "jedit"
  },
  {
    "ident" : "jedit;start_server;xpclient.vmnat.local",
    "objectId" : "xpclient.vmnat.local",
    "values" : [
      true
    ],
    "type" : "ProductPropertyState",
    "propertyId" : "start_server",
    "productId" : "jedit"
  }
]

**productDependency (product dependencies)**

Describes the dependencies of a product to another product as it is defined while creating the package.

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

productOnClient (client specific information to a product e.g. installation state)

Describes which products in which versions are installed on which client.

Example for a productOnClient object:

method productOnClient_getObjects []
{
  "productId": "jedit",
  "clientId": "xpclient.vmnat.local"
}[]
[
  {
    "ident": "jedit;LocalbootProduct;xpclient.vmnat.local",
    "actionProgress": ",",
    "actionDate": "successful",
    "clientId": "xpclient.vmnat.local",
    "modificationTime": "2012-03-30 15:49:04",
    "actionRequest": "none",
    "targetConfiguration": "installed",
    "productVersion": "4.5",
    "productType": "LocalbootProduct",
    "lastAction": "setup",
    "packageVersion": "3",
    "actionSequence": -1,
    "type": "ProductOnClient",
    "installationStatus": "installed",
    "productId": "jedit"
  }
]
**productOnDepot** (depot specific information to a product)

Describes which product is installed in which version on a given depot.

Example for a productOnDepot objects:

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

If you have multiple depot server, you may have different versions of one product.

**config** (administration of host parameter defaults)

Describes the Hostparameter of the opsi-configs Server configuration.

Example for a config object:
method config_getObjects [] {"id":"opsiclientd.event_gui_startup.active"}
[
    {
        "ident" : "opsiclientd.event_gui_startup.active",
        "description" : "gui_startup active",
        "defaultValues" :
            [
                true
            ],
        "editable" : false,
        "multiValue" : false,
        "possibleValues" :
            [
                false,
                true
            ],
        "type" : "BoolConfig",
        "id" : "opsiclientd.event_gui_startup.active"
    }
]

configState (administration of client host parameters)
Describes the Hostparameter of the opsi-configed client configuration.
Example for a configState object:

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

A configState object can’t be created without an existing config object to refer to.
**AuditHardwareOnHost (client specific hardware information)**

Describes the detected hardware types (including the client specific values). The idea is that you can see here the client specific data and in **AuditHardware** only one entry for a network card which is used in all your computers.

Unfortunately in reality this doesn’t work as you might expect.

The attribute **state** describes if this is current (value = 1) or historic (value = 0) data.

Example for a **AuditHardwareOnHost** object:

```json
method auditHardwareOnHost_getObjects []
{"hostId":"xpclient.vmnat.local","hardwareClass":"NETWORK_CONTROLLER","ipAddress":"172.16.166.101"}
[
    {
        "vendorId": "1022",
        "macAddress": "00:0C:29:35:70:A7",
        "hardwareClass": "NETWORK_CONTROLLER",
        "state": 1,
        "deviceType": "PCI",
        "subsystemVendorId": "2000",
        "ipEnabled": "True",
        "type": "AuditHardwareOnHost",
        "firstseen": "2012-03-30 15:48:15",
        "revision": "10",
        "hostId": "xpclient.vmnat.local",
        "vendor": "Advanced Micro Devices (AMD)",
        "description": "Ethernetadapter der AMD-PCNET-Familie",
        "subsystemDeviceId": "1022",
        "deviceId": "2000",
        "autoSense": null,
        "netConnectionStatus": "Connected",
        "maxSpeed": null,
        "name": "Ethernetadapter der AMD-PCNET-Familie",
        "serialNumber": null,
        "lastseen": "2012-03-30 15:48:15",
        "model": null,
        "ipAddress": "172.16.166.101",
        "adapterType": "Ethernet 802.3"
    },
    {
        "vendorId": "1022",
        "macAddress": "00:0C:29:35:70:A7",
        "hardwareClass": "NETWORK_CONTROLLER",
        "state": 0,
        "deviceType": "PCI",
        "subsystemVendorId": "2000",
```
"ipEnabled": "True",
"type": "AuditHardwareOnHost",
"firstseen": "2012-03-08 14:26:14",
"revision": "10",
"hostId": "xpclient.vmnat.local",
"vendor": "VMware, Inc.",
"description": "VMware Accelerated AMD PCNet Adapter",
"subsystemDeviceId": "1022",
"deviceId": "2000",
"autoSense": null,
"netConnectionStatus": "Connected",
"maxSpeed": null,
"name": "VMware Accelerated AMD PCNet Adapter",
"serialNumber": null,
"lastseen": "2012-03-10 14:47:15",
"model": null,
"ipAddress": "172.16.166.101",
"adapterType": "Ethernet 802.3"
},

{
"vendorId": "1022",
"macAddress": "00:0c:29:35:70:a7",
"hardwareClass": "NETWORK_CONTROLLER",
"state": 0,
"deviceType": null,
"subsystemVendorId": "1022",
"ipEnabled": null,
"type": "AuditHardwareOnHost",
"firstseen": "2012-02-29 15:43:21",
"revision": "10",
"hostId": "xpclient.vmnat.local",
"vendor": "Advanced Micro Devices [AMD]",
"description": "Ethernet interface",
"subsystemDeviceId": "2000",
"deviceId": "2000",
"autoSense": "",
"netConnectionStatus": "yes",
"maxSpeed": null,
"name": "79c970 [PCnet32 LANCE]",
"serialNumber": "00:0c:29:35:70:a7",
"lastseen": "2012-03-30 14:58:30",
"model": "79c970 [PCnet32 LANCE]",
"ipAddress": "172.16.166.101",
"adapterType": ""
}
**auditHardware (client independent hardware information)**

Describes the detected hardware types (independent from client specific values). The idea in this object is to see client specific data and in AuditHardware only the generic. That way, for example, you can see here only one entry for a network card, which is used in all your computers.

Unfortunately in reality this idea doesn’t work as you might expect.

Example for a auditHardware object:

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

Describes the detected software types (including the client specific values). The idea is that you will see here the client specific data and in **auditSoftware** only one entry for a office software which is used in all your computers.

Example for a auditSoftwareOnClient object:

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

**auditSoftware (client independent software information)**

Describes the detected software types (independent from the client specific values). The idea is that you will see here only one entry for a office software which is used in all your computers.

Example for a auditSoftware object:
method auditSoftware_getObjects [] { "name": "jEdit 4.5.0" }

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

**auditSoftwareToLicensePool (license management)**

Describes which license pools are assigned to which **auditSoftware** patterns.

Example for a auditSoftwareToLicensePool object:
softwareLicenseToLicensePool (license management)

Describes which softwareLicenseId is assigned to which licensePoolId.

Example for a softwareLicenseToLicensePool object:

```json
method softwareLicenseToLicensePool_getObjects [] {"licensePoolId":"win7-msdn-prof"}
[

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

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

softwareLicense (license management)

Describes the existing software licenses and their meta data.

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

licenseContract (license management)

Describes the existing licenses contracts and their meta data.

Example for a licenseContract object:

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

licenseOnClient (license management)

Describes which license is used by which client.

Example for a licenseOnClient object:
method licenseOnClient_getObjects [] {"clientId":"win7client.vmnat.local"}
[
  {
    "softwareLicenseId": "uib-msdn-win7-vol",
    "ident": "uib-msdn-win7-vol;win7-msdn-prof;win7client.vmnat.local",
    "licenseKey": "12345-12345-12345-12345-3dbv6",
    "notes": "",
    "clientId": "win7client.vmnat.local",
    "licensePoolId": "win7-msdn-prof",
    "type": "LicenseOnClient"
  }
]

licensePool (license management)

Describes the license pool and to which opsi product the license pool is assigned.

Example for a license Pool object:

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

Special methods

Opsi has some special methods. This chapter will introduce some of the more important ones.

configState_getClientToDepotserver

This is in fact also a storage object, but it’s a little aside of the standard. It tells us to which depot a client is currently assigned.

The syntax is
method configState_getClientToDepotserver *depotIds *clientId
*masterOnly *productIds

Example:

```plaintext
method configState_getClientToDepotserver [] "pcbon4.uib.local"
[
   {
      "depotId" : "bonifax.uib.local",
      "alternativeDepotIds" : [
      ],
      "clientId" : "pcbon4.uib.local"
   }
]
```

Communication with hosts

The `hostControl` methods are used to communicate and control the clients. Since opsi 4.0.3 we strongly recommend to use the `hostControlSafe` methods. All `hostControlSafe` or `hostControl` Methods have as last parameter the `hostIds`. The `hostIds` are the list of clients this method should work on. In all `hostControlSafe` methods this parameter is not optional, if you want to send a method to all clients you have to give a "*". In the older `hostControl` methods it is allowed to omit this parameter, which means *send to all*. This has caused some trouble to people which tried this with methods like `hostControl_reboot`. So with opsi 4.0.3 we broke the backward compatibility and now an empty `hostIds` is not any more allowed for the `hostControl_reboot` and `hostControl_shutdown` methods.

- **hostControlSafe_execute**
  Execute a command on the client.
  Connect to the opsiclientd of the given `hostIds` and tell them to start `command`.
  Parameters: `command` `hostIds`

- **hostControlSafe_fireEvent**
  Starts a opsiclientd event on the client.
  Connect the opsiclientd of the given `hostIds` and tell them to start the `event`.
  Parameters: `event` `hostIds`

- **hostControlSafe_getActiveSessions**
  Get information of the logged on users on the client.
  Connect the opsiclientd of the given `hostIds` and ask for the active sessions.
  Parameters: `hostIds`

- **hostControlSafe_opsiclientdRpc**
  Run the web service `method` of the opsiclientd.
  Connect the opsiclientd of the given `hostIds` and tell them to run the web service `method` using the
given parameters. This is the most generic hostControlSafe method, because you may start any possible method. The best way to find out what is possible, is to have a look at control interface https://<clientId>:4441

Parameters: method *params hostIds

- hostControlSafe_reachable
  Checks if the opsiclientd is reachable.
  Connect the opsiclientd of the given hostIds but do not login.
  Parameters: hostIds

- hostControlSafe_reboot
  Reboot the clients.
  Connect the opsiclientd of the given hostIds and starts a reboot.
  Parameters: hostIds

- hostControlSafe_showPopup
  Shows a pop up message on the clients.
  Connect the opsiclientd of the given hostIds and starts a pop up windows with the message.
  Parameters: message hostIds

- hostControlSafe_shutdown
  Shutdown the clients.
  Connect the opsiclientd of the given hostIds and starts a shutdown.
  Parameters: hostIds

- hostControlSafe_start
  Sends a wakeOnLan signal the clients.
  This is the only hostControlSafe method that is not use by the opsiclientd from a client.
  Parameters: hostIds

- hostControlSafe_uptime
  Asks for the clients uptime.
  Connect the opsiclientd of the given hostIds and get the clients uptime in seconds.
  Parameters: hostIds

log_read / log_write

- log_read
  Reads a opsi log file from the server.
  Parameters: logType *objectId *maxSize
  Possible logTypes are instlog (opsi-winst), clientconnect (opsiclientd), userlogin, bootimage, opsiconfd. The objectId is normally the clientId to which the log belongs.

- log_write
  Writes a opsi log file to the server.
  Parameters: logType data *objectId *append
  Logtypes and objectIds see above, append (true/false) (Default = false) should the log be appended to an existing log.
Tutorial: Working with groups

The following tutorial will show how to use the opsi interface from the commandline and work with groups of hosts in opsi.

We want to work with group objects and therefore need to work with those functions whose names start with `group`. Opsi does distinguish between groups of the type `ProductGroup` and `HostGroup`. The first is used for product groupings and the last is used for grouping hosts.

Creating a group of hosts is possible through the method `group_createHostGroup`. The parameters of the method are `id`, a `description`, `notes` and the `parentGroupId` (ID of the parent group). Only the ID is required - everything else is optional. The ID is also the name of the group.

In opsi 4.0 groups are identified by their ID. This ID must be unique throughout the opsi groups.

To create a first group from the commandline we can now issue the following command:

```
opsi-admin -d method group_createHostGroup rechner_wenselowski "Nikos computer"
```

To check if our group was created we use `group_getObjects`.

```
opsi-admin -d method group_getObjects '' '{"id": "rechner_*", "type": "HostGroup"}'
```

To create some hierarchy we need to also specify the ID of the parent group.

```
opsi-admin -d method group_createHostGroup "rechner_wenselowski2" "Undergroup" "" "rechner_wenselowski"
```

We can use the call to `group_getObjects` from earlier to see that our group was indeed created.

Opsi has a default group that behaves like a directory service - i.e. OpenLDAP - that means, that a client can only be member of one group. There is a root group with the ID `clientdirectory` that assumes that exact behavior for any group / client inside. Any client not in a subgroup of `clientdirectory` will be moved to another special group with the ID `NOT_ASSIGNED`. Anyone working with that groups is responsible that clients are not member of multiple subgroups of `clientdirectory`.

Working with the clients is easy now. You probably have noticed that our earlier query to opsi did not show us any signs of clients. That is because the assignment from a client to a group is taken care of another type of object: `objectToGroup`.

To have a client at hand we will first create one:
opsi-admin -d method host_createOpsiClient "wenselowski-test.uib.local"

This client we now want to add to the group we created previously.

opsi-admin -d method objectToGroup_create "HostGroup" "rechner_wenselowski2" "wenselowski-test.uib.local"

Noticed the **HostGroup** as the first parameter? That is again our group type. To check if the creation was successful we can execute the following command:

```bash
opsi-admin -d method objectToGroup_getObjects '' '{"groupType": "HostGroup", "groupId": "rechner_wenselowski2"}''
```

If for some reason we want to remove a client we can do this as well. Just execute the following:

```bash
opsi-admin -d method objectToGroup_delete "HostGroup" "rechner_wenselowski2" "wenselowski-test.uib.local"
```

Finally you may want to clean up the groups we created earlier. The following statements will do this for you:

```bash
opsi-admin -d method group_delete "rechner_wenselowski"
```

### 5.4.2. Action oriented methods

The action oriented methods where introduced in opsi 3. These methods are still available and will still be maintained. Technically methods are mapped to the *object oriented* methods internally.

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

**method authenticated**

Check whether the authentication on the server was successful or not.

**method createClient clientName domain**

Creates a new client.
method createGroup groupId, members=[], description=""

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

method createLicenseKey productId licenseKey

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

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

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

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

Creates a new netBoot (boot image) product.

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

Creates a new product.

method createProductDependency productId action requiredProductId=""
  requiredProductClassId="" requiredAction="" requiredInstallationStatus=""
  requirementType=""

Creates product dependencies.

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

Creates product properties.

method 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 deleteProduct productId
```

Deletes a product from the data base.

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

Deletes product dependencies.

```
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 \textit{opsi-admin}.

\begin{verbatim}
method getBackendInfos_listOfHashes

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

method getClientIds_list

Supplies a list of clients which meet the assigned criteria.

method getClients_listOfHashes

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

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 auditHardwareOnHost_getObjects 
\{ 
  "hostId": \<hostId＞
\}

Supplies the hardware information of the specified computer.

method getHostId hostname

Supplies the hostid of the specified host name.

method getHost_hash hostId
\end{verbatim}
List of properties of the specified computer.

```plaintext
method getHostname hostId
```

Supplies the host name of the specified host id.

```plaintext
method getInstallableLocalBootProductIds_list clientId
```

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

```plaintext
method getInstallableNetBootProductIds_list clientId
```

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

```plaintext
method getInstallableProductIds_list clientId
```

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

```plaintext
method getInstalledLocalBootProductIds_list hostId
```

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

```plaintext
method getInstalledNetBootProductIds_list hostId
```

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

```plaintext
method getInstalledProductIds_list hostId
```

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

```plaintext
method getIpAddress hostId
```

Supplies the IP address of a host.

```plaintext
method getLicenseKey productId, clientId
```

Supplies an available license key of the specified product or the product license key which is assigned to the client.
```plaintext
method getLicenseKeys_listOfHashes productId

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

method getLocalBootProductIds_list

Supplies a list of all known localBoot products.

method getLocalBootProductStates_hash clientIds = []

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

method getMacAddresses_list hostId

Supplies the MAC address of the specified computer.

method getNetBootProductIds_list

Supplies a list of all NetBoot products.

method getNetBootProductStates_hash clientIds = []

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

method getNetworkConfig_hash objectId

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

method getOpsiHostKey hostId

Supplies the pckey of the specified hostid.

method getPcpatchPassword hostId

Supplies the password of pcpatch (encrypted with the pckey of hostId).
```
method `getPossibleMethods_listOfHashes`

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

method `getPossibleProductActionRequests_list`

Lists the available action requests of opsi.

method `getPossibleProductActions_hash`

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

method `getPossibleProductActions_list productId=softprod`

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

method `getPossibleProductInstallationStatus_list`

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

method `getPossibleRequirementTypes_list`

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

method `getProductActionRequests_listOfHashes clientId`

Supplies the list of upcoming actions of the specified client.

method `getProductDependencies_listOfHashes`

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

method `getProductIds_list`

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

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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getServerId clientId</td>
<td>Supplies the opsi-config-server in charge of the specified client.</td>
</tr>
<tr>
<td>getServerIds_list</td>
<td>Supplies a list of the known opsi-config-server.</td>
</tr>
<tr>
<td>powerOnHost mac</td>
<td>Send a WakeOnLan signal to the specified MAC address.</td>
</tr>
<tr>
<td>setGeneralConfig config</td>
<td>Set for client or domain the generalConfig</td>
</tr>
<tr>
<td>setHostDescription hostId</td>
<td>Set a description for a client.</td>
</tr>
<tr>
<td>setHostLastSeen hostId timestamp</td>
<td>Set the <em>last seen</em> time stamp of a client.</td>
</tr>
<tr>
<td>setHostNotes hostId notes</td>
<td>Set the notes for a client.</td>
</tr>
<tr>
<td>setMacAddresses hostId macs</td>
<td>Set the client MAC address in the data base.</td>
</tr>
<tr>
<td>setOpsiHostKey hostId opsiHostKey</td>
<td>Set the <em>pckey</em> for a computer.</td>
</tr>
</tbody>
</table>
method setPcpatchPassword hostId password

Set the encrypted (!) password for hostId

method setProductActionRequest productId clientId actionRequest

Set an action request for the specified client and product.

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

Set an installation status for the specified client and product.

method setProductProperties productId properties

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

method unsetProductActionRequest productId clientId

Set the action request to none.

5.4.3. Backend extensions

opsi 4 brings the feature of a backend extender that allows to extend to core functionality through additional methods that will be served as an extension to the API.

The set of standard API methods will be created through opsicongfd by overlaying the in the .conf-files in /etc/opsi/backendManager/extend.d defined methods.

Backend extensions can be used to implement specific tasks for additional features.

These extensions have to be written as Python code. Extensions are loaded "on to" an BackendManager-instance and can reference it with self.

5.4.4. Accessing the API

The API uses JSON-RPC 1.0 over HTTP for communication. We use basic authentication.

To use this interface POST your calls to the path rpc of your opsi server, i.e. https://opsiserver.domain.local:4447/rpc.
5.5. opsi-backup

5.5.1. Introduction

Like any other system, an opsi-system should also be backed up. Since opsi is a central tool for Windows client and Windows server management, the opsi-server should be backed up. This manual is intended to give an insight into the backup strategy of opsi and also on topics such as restoring and the “DisasterRecovery” of opsi.

5.5.2. Preconditions for a backup

There is not really a prerequisite for backing up an opsi-system. Once you have located the central files and backends of the opsi system, you can save them using various methods. The following instructions should not only answer the question: "What should be backed up?", but also document a way what a backup strategy for an opsi-system could look like.

The backup script should be run as root, either manually or a root cronjob, so that you can read the configuration of opsi and also determine the system configuration. Furthermore, the mysqldump program should be installed for a backup of the mysql backend, which can usually be found in the client packages of mysql.

5.5.3. Quick Start

Creating a backup:

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

Creates a backup of the currently used backends, as well as the configuration files, in the current directory with the name `opsi_backup.tar.bz2`.

Restoring backend data (without configuration files):

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

Restores the data from the backup file `opsi_backup.tar.bz2` in the current directory.

Completely restoring a backup including configuration files:

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

5.5.4. Basic parts of opsi

Opsi can roughly be divided into five parts. The following five parts are specific to opsi and may vary
from system to system depending on the configuration.

**Opsi configuration**

By far the most important part of opsi is the configuration. True to LSB (Linux Standard Base), the configuration of opsi is located under `/etc/opsi`.

This directory mainly contains the backend configuration, the webservice configuration and the SSL certificate for the webservice. Furthermore, backend extensions are stored here, the configuration of `opsipxeconfd`, `opsi-package-updater` with its repositories and also the modules file, which unlocks your co-financed modules.

The directory `/etc/opsi` must be backed up in order to achieve a full recovery after an incident.

This part is secured with `opsi-backup`.

This backup also has another advantage: If you have changed many configuration options of opsi and the system no longer works properly, it is usually easier and quicker to return to a previous working version than troubleshooting.

**Opsi backends**

The opsi backends are listed in the following chapter. These form the heart of the opsi data storage. All clients, products, configurations, statuses, etc... are stored in the respective data storage.

Opsi offers the following data backends:

<table>
<thead>
<tr>
<th>Backend</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file-Backend</td>
<td>File-based backend, currently the default</td>
</tr>
<tr>
<td>mysql-Backend</td>
<td>MySQL-based backend (since opsi 4.0)</td>
</tr>
<tr>
<td>dhcp</td>
<td>Special backend which is used in combination with dhcpd on the opsi-server</td>
</tr>
</tbody>
</table>

If you do not know which backend you are using, you are probably using the file backend. Opsi is also designed to use multiple backends at the same time. Which backends are used for which functions of opsi is configured in `/etc/opsi/backendManager/dispatch.conf`.

This part is secured with `opsi-backup`.

**Opsi depot share**

The depot files are interesting because they contain the actual files of the software to be distributed. The localboot products as well as the netboot products each have their files stored under `/var/lib/opsi/depot`. In previous versions of opsi they were located in the directory `/opt/pcbin/install`. 
Depending on how much software is kept on the opsi-server, and how many operating system installations including drivers are stored, this data volume can take on enormous proportions.

There are different approaches to back up these files. The simplest alternative is \textit{rsnapshot}. However, there are more elegant solutions, such as storing this data in a redundant file system on a SAN, etc.

This part is \textbf{not} backed up by \texttt{opsi-backup}.

\textbf{Opsi workbench}

The opsi workbench area, which is also used as a Samba-share of the same name (\texttt{opsi\_workbench}), contains the state of your own software packaging. By default, the directory is \texttt{/var/lib/opsi/workbench}. If this share is used to store your created packages in different revisions, this directory should also be saved.

The \textit{rsnapshot} tool is also suitable here.

This part is \textbf{not} backed up by \texttt{opsi-backup}.

\begin{itemize}
\item Since opsi 4.1, this directory can be configured individually for each depot server, so that different paths are possible.
\item Before opsi 4.1, this directory was on non-SLES systems located at \texttt{/home/opsiproducts}.
\end{itemize}

\textbf{Opsi repository}

The directory \texttt{/var/lib/opsi/repository} is used to store opsi packages. Unlike the opsi workbench, it is not used to create opsi packages, but the opsi packages that are stored there should not be removed, in order to simplify a possible synchronization with other servers or the synchronization using \texttt{opsi-package-updater}.

These files are not absolutely necessary for a complete recovery, but can also be saved with the \textit{rsnapshot} tool.

This part is \textbf{not} backed up by \texttt{opsi-backup}.

\textbf{TFTP directory}

The TFTP directory contains configuration files for booting via PXE. This directory is located under \texttt{/tftpboot/} on most systems. On SLES this directory is \texttt{/var/lib/tftpboot/opsi/}.

Possibly modified files are e.g. \texttt{linux/pxelinux.cfg/default.menu} or \texttt{linux/pxelinux.cfg/default.nomenu}. These files are created with default values when installing opsi-linux-bootimage. These are not absolutely necessary for disaster recovery.

This part is \textbf{not} backed up by \texttt{opsi-backup}. 
5.5.5. The **opsi-backup program**

The command line program **opsi-backup** is a tool that makes it easy to create and restore backups.

To do this, **opsi-backup** can be controlled with three basic commands: **create**, **restore** and **verify**. The **--help** option gives a detailed overview of all options of **opsi-backup**.

A backup created with **opsi-backup** is a raw backup, which means that no files are backed up at the logical level, but backups of the files stored in the backends are made in the corresponding structures. **Such a backup can therefore only be restored for an identical backend configuration.**

A backup created with **opsi-backup** is always a full backup (**opsi-backup** does not support incremental or differential backups).

It should be noted that **opsi-backup** does not back up depot files, workbench files and repository files. These files should therefore be backed up in a different way.

The backup file created with **opsi-backup** is a compressed tar file, the content of which can be viewed accordingly.

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

A backup created with **opsi-backup** may contain passwords and PC-keys, among other things, and should therefore be stored securely.

### Creating a backup

A new opsi backup is created with the command **opsi-backup create**. If this command is given without further parameters, the program creates an archive with all data of the backends and the configuration. The file name is generated automatically. Additional program help is available for the **opsi-backup create** command, which is output using the **--help** option.

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

It is also possible to specify the file name or the target directory of the new backup. To do this, simply add a file name or a target directory to the corresponding command. If a directory is specified, **opsi-backup** automatically generates a file in this directory. A file name generated by **opsi-backup** has the form `<hostname>_<opsi-version>_<date>_<time>` and is well suited for archiving several backups. If a file name is specified, an older backup with the same name will be overwritten by **opsi-backup**.

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

In addition, the **create** command enables the backup to be controlled using the following options:
- `--backends {file,mysql,dhcp,all,auto}`
  Allows you to select the backends to include in the backup. This option can be specified multiple times to specify multiple backends. The option `--backends=all` includes all backends. The default for this option is `--backends=auto`, which means that opsi-backup tries to determine the backends in use using the configuration file `/etc/opsi/backendManager/dispatch.conf`. Currently, the following backends are supported: `+mysql,file,dhcp`

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

  If you are using an unsupported backend (such as `ldap`), you can convert it with the command `opsi-convert` to a backend which can be backed up with `opsi-backup`.

- `--no-configuration`
  Excludes the `opsi configuration` from the backup.

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

- `-c [{gz,bz2,none}], --compression [{gz,bz2,none}]`
  Specifies the compression method with which the archive is to be compressed. `none` stands for no compression, the default compression is `bz2`.

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

- `--flush-logs`
  The backup of the mysql backend is done internally with a `mysqldump` command. This means that the data is backed up exactly as the database is at that time (regardless of whether the data is already on disk or only in memory). Thus, the backup created may be more up-to-date and different from the state of the database files. If you want to avoid this, the data held in memory by mysql must first be written to disk. If the option `--flush-logs` is specified, opsi-backup will try to perform this operation (i.e. write the data from memory to disks). However, the corresponding database user of the opsi database needs the corresponding MySQL authorization `RELOAD`. By default, the opsi user is created without this right! If the user does not have this right (and the option `--flush-logs` is specified), the backup will fail. Therefore, only use this option if you have previously adjusted the rights of the database user.

  ```bash
  opsi-backup create --backends=mysql --flush-logs
  ```

**Example**
Archiving backups

`opsi-backup` does not have any functionality for archiving backups. The administrator must therefore ensure that created backups are stored securely and versioned. In addition, `opsi-backup` never automatically deletes older backup versions (unless they are overwritten with `create`). Since `opsi-backup` always creates full backups and no incremental backups, the amount of data can quickly grow to a large size. The administrator must thus also ensure that older backups are deleted regularly if necessary.

Verifying a backup

With the command `opsi-backup verify` the archive can be checked for internal integrity. This check is not a logical check of the data, it is a pure check if the data stored in the archive is not corrupted. Additional help is available for the `opsi-backup verify` command, which is shown using the `--help` option.

Example

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

If the `opsi-backup verify` command is explicitly ran on the console, it often makes sense to send the output of `opsi-backup` to the standard output (stdout): `opsi-backup -v verify opsi_backup.tar.bz2`

Listing the contents of a backup

With the command `opsi-backup list` the contents of a backup will be displayed. The listing shows whether configuration data is available, and from which backends data is in the backup.

Example

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

Restoring from a backup file

Restoring an archive is done with the command `opsi-backup restore`. The backends are (by default) imported based on the current configuration. It is therefore not possible to restore a pure backend backup, if an opsi configuration is not available. The command `opsi-backup restore` needs a parameter, which is the backup archive from which data is to be restored. Additional help is available for the `opsi-backup restore` command, which is output using the `--help` option.
opsi-backup restore accepts the following options:

- **--backends {file,mysql,dhcp,auto,all}**
  Specifies the backend to restore. This option may be used multiple times to specify multiple backends. The option **--backends=all** stands for all backends. As a default, the option **--backends=auto** is used, which means that opsi-backup tries to determine which backends to restore from the configuration file /etc/opsi/backendManager/dispatch.conf.

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

  If you changed your backend configuration since you have created the backup, the default setting will not restore up any data.

- **--configuration**
  Restores the opsi configuration. This option is not the default for the restore command.

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

- **-f, --force**
  Before restoring a backup, opsi-backup carries out a systemcheck to see whether the current opsi installation matches the installation of the backup (.opsi version, OS version, host- and domain name). This option can be used to bypass this check.

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

**Example**

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

If a backup is restored to a server and there is no backup of the depot folder, you can use opsi-package-updater and opsi-package-manager to download and install all packages from the repository again. Any changes made to the depot must then be applied again afterwards.

```
opsi-package-updater download --force
opsi-package-manager --install /var/lib/opsi/repository/*.opsi
```

5.6. opsi data storage (backends)
5.6.1. file backend

When using the *file backend* the configuration information is stored in ini-files on the server.

Main features of the backend *file*:

- Current default backend
- The files of this backend are located in `/var/lib/opsi/config`.
- Is implemented on the assumption that the FQDN of the server on which the backend is used corresponds to the FQDN of the config server.

The content and structure of these files is explained in more detail in the Section 5.7.4, “Files of the file backend” chapter.

5.6.2. mysql backend

*mysql backend for inventory data (overview and data structure)*

The hardware and software inventory data are by default stored in text files via the opsi *file backend*. This form of data storage is less suitable for queries and reports. For this, the data can be stored in a SQL database.

Main features of the backend *mysql*:

- Optional (not the default backend)
- For inventory data free of charge, for use with other data you need a paid activation.
- Finely granulated data structure for data storage and additionally a simplified data model for queries.
- A history function, which logs changes to the inventory data.

Due to the very different nature of the hardware components to be inventoried, the data structure is roughly structured as follows:

- A table *host* describes all known clients and provides a unique *host_id*.
- There are two tables for each device type:
  - *HARDWARE DEVICE_* .... describes the device e.g. Network card type with PCI-identifier
  - *HARDWARE_CONFIG ...* describes configuration of the specific network card e.g. MAC address.

The two tables are linked to each other via the *hardware_id* field.

The situation is similar for the software inventory. Again, the *Software* table describes the total software found, while the *Software_Config* table stores the client-specific configuration.

This results in the following list of tables:

- HARDWARE_CONFIG_1394_CONTROLLER
The assignment of the column names to individual device classes results from the following list
(/etc/opsi/hwaudit/locales/en_US):

```plaintext
DEVICE_ID.deviceType = Device type
DEVICE_ID.vendorId = Vendor ID
DEVICE_ID.deviceId = Device ID
DEVICE_ID.subsystemVendorId = Subsystem vendor ID
DEVICE_ID.subsystemDeviceId = Subsystem device ID
DEVICE_ID.revision= Revision
BASIC_INFO.name = Name
BASIC_INFO.description = Description
HARDWARE_DEVICE.vendor = Vendor
HARDWARE_DEVICE.model = Model
HARDWARE_DEVICE.serialNumber = Serial number
COMPUTER_SYSTEM = Computer
COMPUTER_SYSTEM.sku = Stock Keeping Unit
COMPUTER_SYSTEM.systemType = Type
COMPUTER_SYSTEM.totalPhysicalMemory = Physical Memory
COMPUTER_SYSTEM.dellexpresscode = Dell Expresscode
CHASSIS = Chassis
CHASSIS.name = Name
CHASSIS.chassisType = Chassis type
CHASSIS.installDate = Installation date
CHASSIS.serialNumber = Serial number
BASE_BOARD = Base board
BASE_BOARD.product = Product
BIOS = BIOS
BIOS.version = Version
SYSTEM_SLOT = System slot
SYSTEM_SLOT.currentUsage = Current usage
SYSTEM_SLOT.status = Status
SYSTEM_SLOT.maxDataWidth = Maximum data width
PORT_CONNECTOR = Port
PORT_CONNECTOR.connectorType = Attributes
PORT_CONNECTOR.internalDesignator = Internal designator
PORT_CONNECTOR.internalConnectorType = Internal type
PORT_CONNECTOR.externalDesignator = External designator
PORT_CONNECTOR.externalConnectorType = External type
```
PROCESSOR = Processor
PROCESSOR.architecture = Architecture
PROCESSOR.family = Family
PROCESSOR.currentClockSpeed = Current clock speed
PROCESSOR.maxClockSpeed = Maximum clock speed
PROCESSOR.extClock = External clock
PROCESSOR.processorId = Processor-ID
PROCESSOR.addressWidth = Address width
PROCESSOR.socketDesignation = Socket designation
PROCESSOR.voltage = Voltage
PROCESSOR.NumberOfCores = Number Of Cores
PROCESSOR.NumberOfLogicalCores = Number Of Logical Cores
MEMORY_BANK = Memory bank
MEMORY_BANK.location = Location
MEMORY_BANK.maxCapacity = Maximum capacity
MEMORY_BANK.slots = Number of slots
MEMORY_MODULE = Memory module
MEMORY_MODULE.deviceLocator = Device locator
MEMORY_MODULE.capacity = Capacity
MEMORY_MODULE.formFactor = Form factor
MEMORY_MODULE.speed = Speed
MEMORY_MODULE.memoryType = Memory type
MEMORY_MODULE.dataWidth = Data width
MEMORY_MODULE.tag = Tag
CACHE_MEMORY = Cache memory
CACHE_MEMORY.installedSize = Installed size
CACHE_MEMORY.maxSize = Maximum size
CACHE_MEMORY.location = Location
CACHE_MEMORY.level = Level
PCI_DEVICE = PCI device
PCI_DEVICE.busId = Bus id
NETWORK_CONTROLLER = Network adapter
NETWORK_CONTROLLER.adapterType = Adapter type
NETWORK_CONTROLLER.maxSpeed = Maximum speed
NETWORK_CONTROLLER.macAddress = MAC address
NETWORK_CONTROLLER.netConnectionStatus = Net connection status
NETWORK_CONTROLLER.autoSense = auto-sense
NETWORK_CONTROLLER.ipEnabled = IP protocol enabled
NETWORK_CONTROLLER.ipAddress = IP address
AUDIO_CONTROLLER = Audio controller
HDAUDIO_DEVICE = HD Audio device
HDAUDIO_DEVICE.address = Address
IDE_CONTROLLER = IDE controller
SCSI_CONTROLLER = SCSI controller
FLOPPY_CONTROLLER = Floppy controller
USB_CONTROLLER = USB controller
1394_CONTROLLER = 1394 controller
PCMCIA_CONTROLLER = PCMCIA controller
VIDEO_CONTROLLER = Video controller
Examples queries: List of all hard drives:

```
SELECT * FROM HARDWARE_DEVICE_HARDDISK_DRIVE D
LEFT OUTER JOIN HARDWARE_CONFIG_HARDDISK_DRIVE H ON D.hardware_id=H.hardware_id ;
```
The software inventory uses as primary key the following entries:

- **Name**
  This is the `windowsDisplayName` or, if this is not available, the `windowsSoftwareId`. Both values are obtained from the registry:
  - `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall` respectively
  - `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\<id> DisplayName`

- **Version**
  - `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\<id> DisplayVersion`

- **SubVersion**

- **Language**

- **Architecture (32 Bit / 64 Bit)**

In the table `Software_config` these fields are combined to the field `config_id`.

![Database schema: software inventory](image)

**Figure 52. database schema: software inventory**

**mysql backend for configuration data (overview)**

The **mysql backend** for configuration data has been available since opsi 4.0.

This module is currently a co-financed opsi extension. This means it is not free to use. Further details can be found in Section 9.1, “Activation of non-free modules”.

The **mysql backend** has the advantage of a higher performance, especially with large installations.

Here is an overview of the data structure:
Figure 53. database schema: configuration data

Initializing the MySQL-Backend

If the mysql server is not yet installed, this must first be done by:

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

Then a password must be set for the root access of mysql:

```
mysqladmin --user=root password linux123
```
Since MySQL-Server version 5.7 the previously optional strict mode is enabled by default. This causes the opsi-setup --configure-mysql command to fail. Accordingly, the file /etc/mysql/mysql.conf.d/mysqld.cnf should be edited before the command is executed.

The following line must be inserted in the [mysqld] section:

```
sql_mode = NO_ENGINE_SUBSTITUTION
```

Then the service mysql has to be restarted: `systemctl restart mysql.service`.

It is now possible to continue.

The database can now be set up with the command `opsi-setup --configure-mysql`.

An example session:

```
Connecting to host 'localhost' as user 'root'
Successfully connected to host 'localhost' as user 'root'
Creating database 'opsi'
Database 'opsi' created
Creating user 'opsi' and granting all rights on 'opsi'
User 'opsi' created and privileges set
Testing connection to database 'opsi' as user 'opsi'
Successfully connected to host 'localhost' as user 'opsi'
Updating backend config '/etc/opsi/backends/mysql.conf'
Backend config '/etc/opsi/backends/mysql.conf' updated
Initializing mysql backend
```

In the input dialog, all entries can be confirmed with Enter except for the password.

Next you have to specify in `/etc/opsi/backendManager/dispatch.conf` that the mysql backend should also be used. A detailed description of this configuration can be found in the Backend configuration chapter of the getting-started manual. The file itself contains a number of examples of typical
configurations. A configuration for the *mysql backend* (without internal dhcpd) looks like this:

```
backend_.*         : mysql, opsipxeconfd
host_.*            : mysql, opsipxeconfd
productOnClient_.* : mysql, opsipxeconfd
configState_.*     : mysql, opsipxeconfd
.*                 : mysql
```

After completing the configuration, you must execute the following commands to use the now configured and converted backend:

```
opsi-setup --init-current-config
opsi-setup --set-rights
systemctl restart opsiconfd.service
systemctl restart opsipxeconfd.service
```

The *opsiconfd* service has no hard dependency on MySQL by default, since opsi can be used without a MySQL-backend, and the service can also run on another server. Please refer to the documentation of your operating system on how to configure this.

### Manual configuration

Manual configuration can be done via the backend configuration file. By default this is `/etc/opsi/backends/mysql.conf`.

Since python-opsi 4.1.1.76 it has been possible to force the creation of new connections after a certain time in order to avoid problems with timeouts. One indication of such timeouts can be the message *mysql server has gone away*.

You can set a timeout by specifying `connectionPoolRecycling` after how many seconds a new connection should be created. The default value is `-1`, which means no forced reconnection. If this value is set, it should generally be lower than the value for connection timeouts configured on the server (`wait_timeout`).

### Configuration of the MySQL database for external access

The existing database must be configured so that external access is possible, i.e. not only connections from *localhost* are accepted.

Please refer to the manual of your database for the necessary steps.

### 5.6.3. HostControl backend

The HostControl backend does not store any configuration data, but is used to control the opsi clients. This includes, for example, starting clients via Wake-On-LAN or sending control commands to the...
opsi-client-agent.

The configuration of the HostControl backend is done in the configuration file /etc/opsi/backends/hostcontrol.conf. Configuration options are:

- **opsiclientdPort**: Network port for connecting to an opsi-client-agent.
- **hostRpcTimeout**: Timeout (in seconds) when connecting to an opsi-client-agent.
- **resolveHostAddress**: If this option is set to True, when connecting from opsi-server to an opsi-client, the IP address of the client is firstly determined via name resolution. To give preference to the IP address stored in the opsi backend, the option must be set to False.
- **maxConnections**: Maximum number of simultaneous connections to an opsi-client-agent.
- **broadcastAddresses**: Assignment of broadcast addresses and ports to be used for sending Wake-on-LAN broadcasts. Ports can be configured for each broadcast address independently.

### 5.6.4. HostControlSafe-Backend

A peculiarity of opsi 4.0 methods called without parameters is that it matches all existing objects. For example, the command "host_getObjects" without any parameters returns all host objects. This behavior is somewhat problematic in the HostControl backend. Especially with the two commands: hostControl_shutdown and hostControl_reboot. In these cases, calling these methods without parameters would shutdown or restart all clients.

Therefore with opsi release 4.0.3 two changes were introduced:

- The methods hostControl_shutdown and hostControl_reboot are different from the standard opsi 4.0 behavior since this release. These two methods now return an error message when they are called without any parameters.
- A new backend has been introduced (HostControlSafe-Backend), which results in error message for all methods, if they are called without any correct client parameter. To explicity address all clients with a HostControlSafe-Backend method, you can use the * character:

  ```bash
  opsi-admin -d method hostControlSafe_shutdown *
  ```

For the reasons mentioned above, we recommend using hostControlSafe methods on the console or if you are new to using the service methods.
5.6.5. Conversion between different backends

The `opsi-convert` command is used to convert the opsi configuration data between different backends. The destination or source can be assigned in different ways:

- **backend name:**
  By specifying the name, a corresponding backend is specified on the current server. The command `opsi-convert file mysql` converts the data on the current server from the `file backend` to the `mysql backend`.

- **Service address**
  By specifying service addresses, a server can be addressed remotely. The service address has the form `https://<username>@<ipadresse>:4447/rpc`. You will be asked for the passwords. Example:

  ```
  opsi-convert -s -l /tmp/log https://uib@192.168.2.162:4447/rpc
  https://opsi@192.168.2.42:4447/rpc
  ```

- **Configuration directories**
  By specifying a configuration directory for the corresponding backend manager configuration, the source or target can be described in great detail.
       [--log-level {0,1,2,3,4,5,6,7,8,9}] [--clean-destination]
       [--with-audit-data] [-s OLD SERVER ID]
       [--log-file LOGFILE]
       source destination

Convert an opsi database into an other.

positional arguments:
  source                Backend to read data from.
  destination           Backend to write data to.

optional arguments:
  -h, --help            show this help message and exit
  --version, -V         show program's version number and exit
  --quiet, -q           do not show progress
  --verbose, -v         increase verbosity (can be used multiple times)
  --log-level {0,1,2,3,4,5,6,7,8,9}
                        Set log-level (0..9)
  --clean-destination, -c
                        clean destination database before writing
  --with-audit-data, -a
                        including software/hardware inventory
  -s OLD SERVER ID      use destination host as new server
  --log-file LOGFILE, -l LOGFILE
                        Log to this file. The loglevel will be DEBUG.

The backends can either be the name of a backend as defined in
/etc/opsi/backends (file, mysql, ...) or the the url of an opsi configuration
service in the form of http(s)://<user>@<host>:<port>/rpc

5.6.6. Boot files

/tftpboot/linux contains the boot files that are required in conjunction with the PXE network boot.

5.6.7. Securing the shares with encrypted passwords

The opsi-client-agent accesses the shares made available by the opsi-server in order to be able to
install the software on the client.

The system user pcpatch is used for this. Securing these shares and the authentication data of the
pcpatch user are important for: * General system security and data integrity * Protection of licensed
software packages against misuse

In order to provide the opsi-client-agent access to the authentication data, a specific key (opsi-host-key)
is generated for each client when it is created in opsi. This key is stored (in the file backend) in the file
/etc/opsi/pckeys and is transferred to the PC during reinstallation. The transferred key is stored during the installation of the opsi-client-agent in the file c:\program files\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf in such a way that only Administrators have access. Likewise, on the opsi-server only root and members of the opsiadmin group have access to the file /etc/opsi/pckeys. This way, each PC has a key that is only known to the PC and the opsi-server and which is protected against access by normal users. With this key the current password of the system user pcpatch is encrypted on the opsi-server and stored in the backend. This encrypted password is read by the client each time the opsi-client-agent is activated, so that the pcpatch password can be changed at any time and the client is notified of the changed password in an encrypted way.

5.7. Important files on the opsi server

5.7.1. Common configuration files in /etc

/etc/hosts

The IP number and IP name of the clients can be entered here (additional names are aliases, and comments start with the "#" character).

Opsi needs the fully qualified hostname (i.e. including the domain) and this can come from /etc/hosts as well from the DNS.

Example:

```
192.168.2.106  dplaptop.uib.local  dplaptop  # this opsi-server
192.168.2.153  schleppi.uib.local
192.168.2.178  test_pc1.uib.local  # Test-PC PXE-bootprom
```

The output of:

```
getent hosts $(hostname -f)
```

should be similar to:

```
192.168.1.1  server.domain.tld  server
```

If the result does not look like this (e.g. contains 127.0.0.1 or localhost), you must first correct your /etc/hosts or name resolution.

/etc/group

Two groups must exist here: pcpatch and opsiadmin. All users who are dealing with package management should be member of the group pcpatch. All users who want to use the opsiconfd web service, e.g. via opsi-configed, must be in the opsiadmin group.
/etc/opsi/backends/

Configuration files for the used backends.

/etc/opsi/backendManager/

- acl.conf
  Configuration of the access rights to the opsi methods. This allows access rights for the basic methods of the web service to be restricted to certain users and certain attributes.

- dispatch.conf
  Configuration of which of the backends configured under /etc/opsi/backends/ should be used for what.

- extend.d/
  Directory for backend extensions. So here are for example the scripts that map the opsi 3 methods to the opsi 4 methods.

/etc/opsi/hwaudit/*

Since opsi version 3.2

Here you will find the configuration files for the hardware inventory.

Translations are located in the locales directory.

The mapping between WMI classes and the opsi data storage is configured in the file opsihwaudit.conf.

/etc/opsi/opsi.conf

Since opsi version 4.0.2-2

General opsi settings.

Example:

```ini
[groups]
fileadmingroup = pcpatch
```

Background: The classic installation variant with the user: pcpatch with the primary group: pcpatch does not work with Samba 4. Since Samba 4 is subject to the basic restrictions of Active Directory, groups with the same name as users (as is usual in Unix/Linux) are no longer allowed. For this reason, a new configuration file has been introduced: /etc/opsi/opsi.conf, which controls how the group is determined for Samba access to the shares. In the case of Samba 4 installations, the group name pcpatch is now renamed via this file and is now called opsifileadmins. This means that the users which must have access rights to the shares of opsi (opsi-packagers) under Samba 4 cannot become a member of the pcpatch group, but must be a member of the opsifileadmins group.
/etc/opsi/modules

Since opsi version 3.4

This file is signed byuib gmbh for the activation of non-free features. If this file is changed, it loses its validity. Without this file, only the free features are available.

Directory /etc/opsi/modules.d/

Since opsi version 4.1

Directory reserved for future usage.

/etc/opsi/opsiconfd.conf

Since opsi version 3.0

Configuration file for the opsiconfd service in which settings such as ports, interfaces and logging are specified.

/etc/opsi/opsiconfd.pem

Since opsi version 3.0

Configuration file for the opsiconfd service in which the ssl certificate is stored.

/etc/opsi/opsipxeconfd.conf

Configuration file for the opsipxeconfd service, which is responsible for writing the startup files for the Linux bootimage. Directories, defaults and log levels can be configured here.

/etc/opsi/opsi-package-updater.conf

Configuration file for the opsi-package-updater.

See also Section 5.3.3, “Tool: opsi-package-updater”

5.7.2. Boot files

Boot files in /tftpboot/linux

- pxelinux.0
  Boot file that is loaded in the first step by the PXE boot.

- install and miniroot.gz
  Installation bootimage, which is transferred via tftp to the client during reinstallation.

Boot files in /tftpboot/linux/pxelinux.cfg

- 01-<mac adress> or <IP-address-in-hex>
Files with the hardware address of the client and the prefix 01- can be found on the opsi-server as client-specific boot files. Usually they are created via opsipxeconfd as named pipes and should initiate a reinstallation of the client.

- **default**
  The file default is loaded if there are no client-specific files. If this file is loaded, the client then continues to a local boot.

- **install**
  Information for the boot of the installation bootimage, which is written from opsipxeconfd into the named pipe.

### 5.7.3. Files in /var/lib/opsi

**/var/lib/opsi/depot**

This directory is exported (read-only) as the Samba share opsi_depot. In old opsi installations this directory was /opt/pcbin/install. If this directory still exists, a symlink references it to /var/lib/opsi/depot.

**/var/lib/opsi/ntfs-images**

This is the default directory where the partition images are stored, that are used with the netboot product opsi-clonezilla.

**/var/lib/opsi/repository**

This is where opsi-product-packages are saved, which are installed onto the server by executing opsi-package-updater.

This is also where opsi-product-packages, which are installed by calling the opsi-package-manager, if this is called with the option -d, are saved.

**/var/lib/opsi/workbench**

This location contains the state of your own software packaging.

**Other directories**

The remaining directories in /var/lib/opsi (config and audit) are directories of the file backend, which is described in the following chapter.

### 5.7.4. Files of the file backend

**/etc/opsi/pckey**

The client-specific opsi-host-keys and also the server key are stored here.

Example:
/etc/opsi/passwd

The passwords encrypted with the server key (e.g. for pcpatch) are stored here.

Overview of /var/lib/opsi

The files of the file backend of opsi 4 can by default be found in /var/lib/opsi/config/. The following diagram gives an overview of the directory structure:

```
/var/lib/opsi-
  |-depot      opsi_depot share
  |-repository opsi package repository used by opsi-package-updater
  and opsi-package-manager
    |-audit    inventory files
     !-config/-| config share
       |-clientgroups.ini client groups
       |-config.ini Host Parameters (Global Defaults)
       |-clients/ <pcname.ini> files
       |-products/ product control files
     !-depots depot description files

+audit/
  global.<Type> (generic hard-, and software information)
  <FQDN>.<Type> (host specific hard-, and software information)

clientgroups.ini (contains the hostgroups)

+clients/
  <FQDN>.ini (client configuration information)
  config.ini (contains the host specific host parameters)

+depots/
  <FQDN>.ini (Information of the depots)

+products/
  <ID>_<ProdVer>-<PackVer>.<Type> (Information about the products)

+templates/
  pcaproto.ini (template for new clients)
  <FQDN>.ini (template for specific new clients)
```
Configuration files in detail

The following chapters explain the structure of the different configuration files of the file backend.

./clientgroups.ini

This file contains information about the client groups.

```
[<GroupId>]
<HostId> = 1 #active
<HostId> = 0 #non-active
```

./config.ini

Here you will find the default values of the server configuration as shown in `opsi-configed` in the `host parameters` tab.

./clients/<FQDN>.ini

The client-specific configurations are stored in this file. The information is combined with the values from `<depot-id>.ini`, whereby the settings from `<FQDN>.ini` take precedence.

These files are structured as follows:

The `info` section contains all information directly related to the client, for example the description:

```
[info]
description = <String>
created = <Date> #format: 'YYYY-MM-DD HH:MM:SS'
lastseen = <Date> #format: 'YYYY-MM-DD HH:MM:SS'
inventorynumber = <String>
notes = <String>
hardwareaddress = <MAC> #format: 'hh:hh:hh:hh:hh:hh'
ipaddress = <IP> #format: 'nnn.nnn.nnn.nnn'
onetimepassword = <String>
```

The following section stores the current status of the products on the client. If there are no entries, `not_installed: none` is assumed.

```
[<Type>_product_states] #'LocalbootProduct', or 'NetbootProduct'
<ProductId> = <InstallationStatus>:<ActionRequest>
```

More detailed information can be found in the sections belonging to the respective products:
5. opsi-server

```python
[<ProductId>-state]
producttype = <Type> # 'LocalbootProduct', or 'NetbootProduct'
actionprogress = <String>
productversion = <ProdVer>
packageversion = <PackVer>
modificationtime = <Date> # format: 'YYYY-MM-DD HH:MM:SS'
lastaction = <ActionRequest>
actionresult = <ActionResult>
targetconfiguration = <InstallationStatus>
```

/var/lib/opsi/config/templates

This is the location of the file `pcproto.ini`, which is the standard template for creating new client ini-files and has the same structure. If specific clients should be given different information, you can save a `<FQDN>.ini` in this directory.

/var/lib/opsi/config/depots/

Here you will find the files of the `opsi-depot-server`, which are also saved with `<depot-id>.ini`. Among other things, the connection to the depot is stored here.

```ini
[depotshare]
remoteurl = smb://<NetBiosName>/<Path>
localurl = file://<Path>

[depotserver]
notes = <String>
network = <IP>
description = <String>
hardwareaddress = <MAC>
ipaddress = <IP>
inventorynumber = <String>

[repository]
remoteurl = webdavs://<FQDN>:<Port>/<Path>
localurl = file://<Path>
maxbandwidth = <Integer> # in Bytes
```

Here you will also find information on which opsi products, in which version, and with which property defaults are installed on the depot.

**Product control files in /var/lib/opsi/config/products/**

The product control files contain the metadata of the products, e.g. name, properties and their default values, dependencies ...
The control files correspond to the control files that are created when creating opsi products in the directory `<product name>/OPSI/control`.

The control files consist of the following sections:

- **Section [Package]**
  Description of the package version and packages required for the installation of the package on an opsi-depot-server.

- **Section [Product]**
  Description of the product.

- **Section(s) [ProductProperty]**
  (optional)
  Description of editable product properties.

- **Section(s) [ProductDependency]**
  (optional)
  Description of product dependencies.

Example:
[Package]
version: 1
depends:

[Product]
type: localboot
id: thunderbird
name: Mozilla Thunderbird
description: Mail client by 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 plug-in 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, for instance, distinguish packages with the same product version but with a different opsi-winst script.

- [Package]-depends
  Specifies a package required for installation on an opsi-depot-server. Specific versions can be configured by specifying the version in parenthesis after the package name. One of the following operators must precede the version within the parenthesis: =, <, <=, >, >=.

- [Package]-Incremental
  This is an obsolete, setting that has no effect, which has not been set in new packages since opsi 4.1. You can remove this entry.

- [Product]-type
  Specifies the product type as localboot or netboot.

- [Product]-Id
Is a unique identifier for the product, independent of the version.

- [Product]-name
  Is the full name of the product.

- [Product]-Description
  Is an additional description of the product, which is for example shown in opsi-configed under Description.

- [Product]-Advice
  is an additional description, for the handling of the product (usually), which is shown in opsi-configed under Note.

- [Product]-version
  Is the version of the packaged software.

- [Product]-Priority
  Influences the installation sequence, in combination with the product dependencies.

- [Product]-productClasses
  Is not currently used (nor is it displayed).

- [ProductProperty]-type
  Type of the property: (unicode/boolean)

- [ProductProperty]-name:
  Name of the property.

- [ProductProperty]- multivalue
  Can this property contain a list of values. (True/False)

- [ProductProperty]- editable
  Can this property be freely edited (or can only a value from a predefined list be selected). (True / False)

- [ProductProperty]-description:
  Description of the property (tooltip in opsi-configed).

- [ProductProperty]-values :
  List of possible, allowed values. If empty, the value is freely editable.

- [ProductProperty]-default :
  Default value of the property.

- [ProductDependency]-Action :
  For which action of the product you are currently creating, should the dependency apply (setup, uninstall ...).

- [ProductDependency]-Requiredproduct:
  Product-id (identifier) of the product to which a dependency exists.

- [ProductDependency]-Required action:
  You can either request an action or (see below) a status. Actions could be: setup, uninstall, 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 is not installed on the client yet.

- **[ProductDependency]-Requirement type:**
  Installation order. If the product to which a dependency exists must be installed before the current product can be installed, then this is set to before. If it has to be installed after the current product, this is set to after. If the order is irrelevant, nothing has to be entered here.

**Inventory data /var/lib/opsi/audit**

This is the location of the inventory data for hardware (*.hw) and software (*.sw).

### 5.7.5. opsi programs and libraries

**Programs in /usr/bin**

- **opsipxeconfd**
  Opsi daemon, which manages the files in the tftp area of the server that are necessary for the PXE boot of the clients.

- **opsi-admin**
  Command line interface to the opsi python library.

- **opsiconfd**
  Opsi daemon that provides the opsi methods as a webservice and much more.

- **opsiconfd-guard**
  Opsi daemon that monitors whether the **opsiconfd** is running and restarts it if necessary.

- **opsi-configed**
  Command to start the opsi management interface.

- **opsi-convert**
  Script for converting between different backends.

- **opsi-makepackage**
  Script for packing the opsi-package (opsi-product)

- **opsi-newprod**
  Script for creating a new product.

- **opsi-package-manager**
  Script to install and uninstall opsi packages on an opsi server.

- **opsi-setup**
  Program for various basic settings.

### 5.7.6. opsi log files

The opsi log files have the following format:
The log levels are:

0 = nothing      (absolute nothing)
1 = essential    ("we always need to know")
2 = critical     (unexpected errors that may cause a program abort)
3 = error        (Errors that will not abort the running program)
4 = warning      (you should have a look at this)
5 = notice       (Important statements to the program flow)
6 = info         (Additional Infos)
7 = debug        (important debug messages)
8 = debug2       (a lot more debug information and data)
9 = confidential (passwords and other security relevant data)

/var/log/opsi/bootimage

Here you can find the log files of the bootimage from the clients. These files are created as <fqdn>.log.

If the bootimage can not connect to the web service, the log file can be found on the client under /tmp/log. In such a case, there are two ways to get the log file from the client:

1. You have a network connection to the client
   Then you can use SCP to get the file from /tmp/log. For Windows you can use for example WinSCP.

2. You have no network connection to the client
   Then a USB stick can help:
   - Log in as root with password linux123
   - Insert the USB stick and wait a few seconds
   - Check with sfdisk -l which device the stick is
   - mount
   - copy
   - unmount

An example session:
#sfdisk -l
Disk /dev/sda: 30401 cylinders, 255 heads, 63 sectors/track
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0

Device Boot Start   End     #cyls  #blocks   Id  System
/dev/sda1   *      0+  30401-  244197528+   7  HPFS/NTFS
/dev/sda2          0       -       0          0    0  Empty
/dev/sda3          0       -       0          0    0  Empty
/dev/sda4          0       -       0          0    0  Empty

Disk /dev/sdb: 1017 cylinders, 33 heads, 61 sectors/track
Units = cylinders of 1030656 bytes, blocks of 1024 bytes, counting from 0

Device Boot Start   End     #cyls  #blocks   Id  System
/dev/sdb1          0+   1016    1023580    b  W95 FAT32
/dev/sdb2          0       -       0          0    0  Empty
/dev/sdb3          0       -       0          0    0  Empty
/dev/sdb4          0       -       0          0    0  Empty

# mount /dev/sdb1 /mnt
# cp /tmp/log /mnt
# umount /mnt

/var/log/opsi/clientconnect

This is the location of the log files of the opsi-client-agent running on the clients.
This is C:\opsi.org\log\opsiclientd.log on the client.

/var/log/opsi/instlog

This is the location of log files of the opsi-winst scripts executed on the clients.
The originals are on the client at C:\opsi.org\log\opsiscript.log

/var/log/opsi/opsiconfd

This is the location of log files of opsiconfd itself as well as log files of the clients.
The files are created as <IP address>.log and if configured in /etc/opsi/opsiconfd.conf, symbolic links
for these are created as <fqdn>.log.

/var/log/opsi/opsipxeconfd.log

Log file the opsipxeconfd
which manages the files in the tftp area of the server that are necessary for the PXE boot of the clients.

/var/log/opsi/package.log

Log file of opsi-package-manager.
5.8. Upgrade instructions for opsi-server

As of opsi 4.0, you will find these in the version-specific release notes manuals.

5.9. Special notes on UCS 4.X systems

UCS 4.X requires a DHCP policy to enable a PXE boot from a UCS system. This policy must be made in the domain policy settings for DHCP boot. For the bootserver option, you enter the server on which the boot file is located. The boot file name is pxelinux.0. Since this file is located directly in the directory /tftpboot/, the entry will look like this: linux/pxelinux.0

5.10. Scheduling installations

5.10.1. The cronjobs wake_clients_for_setup.py, opsi-auto-update and working_window

With the opsi product opsi-auto-update devices can be updated easily and safely without having to set the update for each individual device. The product opsi-auto-update is described here: Section 8.1.1.19, “opsi-auto-update”.

With the help of cron jobs on the opsi-config-server, the execution of opsi products can be timed and thus e.g. postponed until night.

The prerequisite for this is that the clients can be woken up via wake-on-lan (WOL) or started on a...
certain time via BIOS.

In order to make the control via cron job as simple and effective as possible, we have developed the script `wake_clients_for_setup.py`. This script has the following tasks:

- For a specific group of clients
- a certain group of products is set to `setup`
- The selected clients are then woken up via `wake-on-lan`
- If the clients were only `sleeping` and were not booted, the clients can still be sent the signal to execute a certain event.

The selected clients can be specified either:

- by specifying a `host group`, which can for example be created with opsi-configed (see: Section 4.7, “Client selection and hierarchical groups using the tree view”)
  
  ```
  --host-group-id HOSTGROUPID
  ```

- by specifying an opsi-depot (all clients of the depot are treated)
  
  ```
  --depotId DEPOTID
  ```

- by specifying a file in which the clients are listed
  
  ```
  --host-file INPUTFILE
  ```

The specification of the selected products which are set to `setup` is done by specifying a productgroup, which can for example be created with opsi-configed (see: Section 9.17.3.1, “Managing product-groups”)

```
--product-group-id PRODUCTGROUPID
```

The event to be triggered is specified via the parameter `--event EVENTNAME`

The names of groups in opsi must be `unique`. Regardless of whether it is a hostgroup from the `Directory` or `Groups`, or a productgroup: A group name may only appear once.

Here is an overview of the parameters of `wake_clients_for_setup.py`:

```
# wake_clients_for_setup.py --help
usage: wake_clients_for_setup.py [-h] [--version] [--verbose]
                                [-log-level {0,1,2,3,4,5,6,7,8,9}]
                                [-wol-timeout WOLTIMEOUT]
                                [-ping-timeout PINGTIMEOUT]
                                [-connect-timeout CONNECTTIMEOUT]
                                [-event-timeout EVENTTIMEOUT]
                                [-reboot-timeout REBOOTTIMEOUT]
                                [-host-group-id HOSTGROUPID]
                                [-depotId DEPOTID] [--host-file INPUTFILE]
                                [--product-group-id PRODUCTGROUPID]
```
Wake clients for software installation.

optional arguments:
-h, --help            show this help message and exit
--version, -V         show program's version number and exit
--host-group-id HOSTGROUPID, -H HOSTGROUPID
    Group in which clients have to be to be waked up.
    (default: None)
--depotId DEPOTID, -D DEPOTID
    DepotId in which clients have to be registered to be
    waked up. (default: None)
--host-file INPUTFILE, -F INPUTFILE
    Filename with clients per line have to be waked up.
    (default: None)
--product-group-id PRODUCTGROUPID, -P PRODUCTGROUPID
    ID of the product group to set to setup on a client
    (default: None)
--event EVENTNAME, -E EVENTNAME
    Event to be triggered on the clients (default: None)
--reboot, -X          Triggering reboot on the clients (default: False)
--no-auto-update NOAUTOUPDATE, -N NOAUTOUPDATE
    Do not use opsi-auto-update product. (default: False)

Logging:
--verbose, -v         increase verbosity on console (can be used multiple
    times) (default: 4)
--log-level {0,1,2,3,4,5,6,7,8,9}, -l {0,1,2,3,4,5,6,7,8,9}
    Set the desired loglevel for the logfile. (default: None)

Timeouts:
--wol-timeout WOLTIMEOUT
    Time to wait until opsiclientd should be reachable.
    (default: 300)
--ping-timeout PINGTIMEOUT
    Time to wait until client should be pingable.
    (default: 300)
--connect-timeout CONNECTTIMEOUT
    Timeout for connecting to opsiclientd. (default: 15)
--event-timeout EVENTTIMEOUT
    Time to wait until opsiclientd should be processing.
    (default: 300)
--reboot-timeout REBOOTTIMEOUT
    Time to wait before opsiclientd will be reboot the
    client. (default: 60)
The script `wake_clients_for_setup.py` can be found here:  
https://download.uib.de/opsi4.1/misc/time-driven-maintenance-tools/wake_clients_for_setup.py

Save it in `/usr/local/bin` on the opsi-server and make it executable:

```
chmod u+x /usr/local/bin/wake_clients_for_setup.py
```

Execution would for example be the following:

```
# wake_clients_for_setup.py --event=gui_startup --product-group-id=nightly-cron-products
# --host-group-id=nightly-cron-group-0

```

The clients of the host group `nightly-cron-group-0` are selected and the products of the product group `nightly-cron-products` are set to `setup`. Then the selected clients are woken up using `wake-on-lan`, and after a short period the command is sent to them to execute the event `gui_startup`.

To execute this daily at a certain time, this command must be entered in the `crontab` of the server. For example, the command `crontab -e` can be used (as root).

In the crontab there is a time specified before the command. This consists of 5 parts, of which only the first two interest us: minute, hour.

A crontab in which different client groups are called up during the night is shown in the following example:
# For more information see the manual pages of crontab(5) and cron(8)

# m h  dom mon dow  command

# cronjobs to wake up and update the PCs
5 0 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-cron-group-0 --wol
--timeout=120 --event-timeout=120
30 0 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-jpr-group-030 --wol
--timeout=120 --event-timeout=120
59 0 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-cron-group-1 --wol
--timeout=120 --event-timeout=120
30 1 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-jpr-group-130 --wol
--timeout=120 --event-timeout=120
5 2 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-cron-group-2 --wol
--timeout=120 --event-timeout=120
30 2 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-jpr-group-230 --wol
--timeout=120 --event-timeout=120
5 3 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-cron-group-3 --wol
--timeout=120 --event-timeout=120
30 3 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-jpr-group-330 --wol
--timeout=120 --event-timeout=120
5 4 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-cron-group-4 --wol
--timeout=120 --event-timeout=120
30 4 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-jpr-group-430 --wol
--timeout=120 --event-timeout=120
5 5 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-jpr-group-500 --wol
--timeout=120 --event-timeout=120
35 5 * * * /usr/local/bin/wake_clients_for_setup.py --log-level=5 --event=gui_startup
--product-group-id=nightly-cron-products --host-group-id=nightly-jpr-group-530 --wol
--timeout=120 --event-timeout=120

It is possible to prevent installations from accidentally happening outside the planned maintenance window. In a school, for example, when a student turns on a computer during the day, it should be immediately available and therefore no installations are to be carried out, even if action requests are set. A working window can be set in the configuration of the opsiclientd for certain events (usually
gui_startup).
How this working_window is configured is described here: Section 6.1.3.8, “Working Window”
6. opsi-client

6.1. opsi-client-agent

6.1.1. Overview

In order that the distribution of software does not become unmanagable, a client PC has to notice independently that new software packages or updates are available and install them. Any form of user interaction is to be avoided during the installation, so that this can take place unattended and the necessary installations are not interrupted by uncertain users.

With opsi, these requirements are met by an agent on the client:

The so-called opsi-client-agent is installed on the client. This usually checks at the start of the client and before the user logs in, based on configuration information on the opsi-config-server, whether an update should be installed for this client.

If software is to be installed, the script-controlled installation program opsi-winst is started. The necessary scripts and software packages are available on a file share, the so-called opsi-depot. During this time, there is no need and no possibility for the user to intervene in the installation process.

To prevent a user from logging on to the system before completing the installation and thus disrupting the installation process, the so-called opsi-login-blocker is also installed, which only allows logon after the installation has been completed.

In order for software packages to be installed without interaction with the program opsi-winst, they have to be prepared for this. See the chapter Integrating new software packages in the software distribution of opsi in the opsi-getting-started manual.

6.1.2. Directories of the opsi-client-agent

The opsi-client-agent is installed in %ProgramFiles%\opsi.org\opsi-client-agent.

This directory contains all programs of the opsi-client-agent such as opsiclientd, opsiclientd notifier, opsi-winst and some libraries. The configuration files and graphical templates of the mentioned programs can also be found here.

The directory %ProgramFiles%\opsi.org\opsi-client-agent is protected against changes by users without administrative privileges.

The directory %ProgramFiles%\opsi.org\opsi-client-agent\opsiclientd contains the configuration file of opsiclientd and can only be accessed with administrative rights.

Furthermore, there is the directory c:\opsi.org.

This directory is currently used for caching installation files and data (see WAN extension). This directory will have additional functions in the future.

The directory c:\opsi.org can only be accessed with administrative rights.
Log files of the opsi-client-agent are located in `c:\opsi.org\log`.

### 6.1.3. The service: opsiclientd

opsiclientd is the basis of the opsi-client-agent. It runs as a service with administrative rights and is started automatically at boot.

The main functions are:

- Event-based control: There can be reacted to different events on the system. An event is, for example, the start of the operating system.
- Control via web service: The web service of opsiclientd can be accessed via the network. This interface is used to initiate installations (push) but also for maintenance purposes.
- Remote configuration: All essential configuration data of opsiclientd can be edited globally or client-specifically via the host parameter.

The opsi-client-agent consists of several components:

- opsiclientd: The central service of the opsi-client-agent.
- opsiclientd notifier: Window for information / communication with the user
- opsi-login-blocker: Blocks the login until the installations are completed.

### Installation

As part of a new installation of an operating system via unattended setup with opsi, the opsi-client-agent is automatically installed.

To uninstall the opsi-client-agent, the action request can be set to `uninstall`.

For installation afterwards or for repair purposes, see chapter Section 6.1.6, “Subsequent installation of the opsi-client-agents”.

### opsiclientd

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

The opsiclientd has the following tasks:

- while the system is booting and the opsiclientd is waiting for the GUI to come up, the block_login_notifier is started and shows a padlock at the right upper corner of the screen.
- Getting in action if the configuration event takes place. In case of action the opsiclientd contacts the opsi server via web service (JSON-RPC) and asks for the configuration data and required actions. The default event is gui_startup which will fire at boot time before user login.
- Creates a named pipe which is used by the opsi-login-blocker to ask via JSON-RPC the opsiclientd when to unblock the login.
• Starting the `opsiclientd notifier` as a thread for information and interaction with the user.
• If needed, it connects to the `opsi-depot` to update the local installation of the `opsi-winst` and then starts it to process the `action requests` (software packet installations).

**opsiclientd notifier**

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

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

**blocking notifier**

Indicates that the `opsi-login-blocker` is blocking

![Figure 56. opsiclientd blocklogin notifier](image)

**event notifier**

Shows information about the current event.

![Figure 57. opsiclientd event notifier](image)

**action notifier**

Shows state of the event processing.

![Figure 58. opsiclientd action notifier](image)
**shutdown notifier**

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

![shutdown notifier](image)

*Figure 59. opsiclientd shutdown notifier*

Names and functionality of the notifier have changed from opsi 4.0 to opsi 4.0.1.
The opsi 4.0 event notifier doesn’t exist anymore.
The opsi 4.0.1 event notifier equals the opsi 4.0 action notifier.
The opsi 4.0.1 action notifier has almost the same functionality as the opsi 4.0 event
notifier, but it will only be activated if there is a *action request*.

**opsi-login-blocker**

The *opsi-login-blocker* for NT5 (Win2K/WinXP) is implemented as a GINA (`opsigina.dll`). This GINA
waits until the opsiclientd reports, that all *product actions* are finished or, if the opsiclientd is not
reachable, until the connection timeout to the opsiclientd is reached (normally 120 seconds). Then the
complete control is forwarded to the next GINA, which is normally the `msgina.dll`.

The *opsi-login-blocker* for NT6 (Vista/Win7) is implemented as a *credential provider filter*
(`OpsiLoginBlocker.dll`). This *credential provider filter* blocks all *credential providers* until the
opsiclientd reports, that all *product actions* are finished or, if the opsiclientd is not reachable, until the
connection timeout to the opsiclientd is reached (normally 120 seconds).

**Processing sequence**

How the opsiclientd works may be configured in many details. To understand these configuration
options, it is necessary to understand the processing sequence. Here comes an overview of the work
flow of a *standard event* like the `event_gui_startup`.
Figure 60. simplified work flow of a standard event

The most important parameters have the following relations:

- If there is an error while connecting to the *opsi-config-server*, the log of this problem cannot be sent to the server. But you may find the log in the local log file `opsiclientd.log` in the log directory (`c:\opsi.org\log\opsiclientd.log`) at the client.

1. If an event fires, the `event_notifier_command` will be started.
   Now the opsiclientd tries to reach the *opsi-config-server* using the url address.
   If after `userCancelableAfter` seconds there is still no connection established, so the *opsiclientd notifier* will enable an *Abort* button. If no connection could be established in `connectionTimeout` seconds, the *opsiclientd* connection process will be aborted and the event ends with an error message. To avoid a user from aborting, set `userCancelableAfter = connectionTimeout`.
2. After a successful connection to the opsi-config-server, the opsiclientd checks if there are action requests for this client. If there are action requests and the action_warning_time > 0, the action_notifier_command will be executed. This is normally the opsiclientd notifier, which shows now the list of action requests for this client for action_warning_time seconds.

Is the action_warning_time = 0 (default) the action_notifier_command will not be executed. You may allow the user to suspend the process at this time by setting action_userCancelable >= 0. The user may suspend the actions up to action_userCancelable times. After action_userCancelable aborts in sequence or if action_userCancelable = 0 the user gets no possibility to suspend the actions.

In every case there will be a button which allows the user to start the installations immediately without waiting for the count down of action_warning_time seconds. The messages displayed by the opsiclientd notifier may be configured with the options action_message or action_message[lang]. This messages may contain the placeholders %action_userCancelable% (total number of allowed suspensions) and %action_cancelCounter% (number of suspensions already used by the user).

If the actions are not suspended by the user, the action_cancelCounter will reset and the opsi-winst will be executed to process the action requests.

3. If the opsi-winst terminates with a reboot or shutdown request, the shutdown_notifier_command will be executed if shutdown_warning_time > 0.

The now starting shutdown_notifier_command shows for shutdown_warning_time seconds a message saying that the client will be rebooted. If shutdown_userCancelable > 0 the user may suspend the reboot up to shutdown_userCancelable times in sequence. If the user suspends the reboot, the shutdown_notifier_command will be restarted after shutdown_warning_repetition_time. The shutdown_notifier_command shows a message which may be configured by shutdown_warning_message or shutdown_warning_message[lang]. This message may contain the placeholders %shutdown_userCancelable% (maximum number of allowed suspensions) and %shutdown_cancelCounter% number of suspensions already done by the user).

If the client is rebooted (by the user or the opsi-client-agent) the %shutdown_cancelCounter% will be reset.

The sequence of event processing and user actions is visualized as a timeline graphic at the info page of the opsiclientd.

(Section 6.1.3.14, “opsiclientd infopage”).

Configuration

The following chapters shows how to configure the opsi-client-agent.

Configuration of different events

To meet the requirements of the various different situations in which the opsi-client-agent will become active, a slightly complex configuration is needed. To reduce the complexity, the configuration file uses something like inheritance.

In the opsiclientd configuration section headers like [event_<config-id>] introduce a new event configuration section. An event configuration may be disabled by setting the section option active =
false.

There are different types of event configurations (type).

- There are event configuration templates (type = template).
  Event configurations may inherit configurations from another event. In this case the option super points to the other event to inherit all parameters from (excluding the parameter active). These inherited parameters may be overridden by local parameters in the current event section. So an event section needs only those parameters which are different from the super event.
  Setting an event to active = false does not change anything in the inheritance process.

- The other event types are:
  - gui startup
    A gui startup event starts while booting the client and loading the graphical user interface (GUI). It is the most used event and set to active in the default configuration.
  - custom
    Event configurations of the type custom are fired by a wql event. A wql event is defined by the corresponding wql statement in the event configuration. If the wql statement is empty, the event will never be fired, but can be executed from the interactive web interface.
  - user login
    will be fired at the login of a user
  - timer
    will be fired all interval seconds
  - sync completed
    will be fired if the synchronization of configurations (sync_config_from_server) or products (cache_products) is completed.
  - sw on demand
    will be fired by the user choosing Start actions now in the software-on-demand web page of the opsiclientd. It will never be fired if software-on-Demand is not used.

- There are Preconditions
  Preconditions define special system states (e.g. a user is logged on). In the opsiclientd configuration a section header of the form [precondition_<precondition-id>] starts the declaration of a Precondition. A Precondition is true, if all declared options are true. An option not declared (but possible) is assumed as true.
  Possible options for Preconditions are:
  - user_logged_in: is true if currently a user is logged on.
  - config_cached: is true if the caching of configuration data is completed (see: sync_config_from_server).
  - products_cached: is true if the caching of product files is completed (see: cache_products).

- A Precondition can be assigned to an event configuration.
  This can be done by giving the precondition in curly brackets at the end of the event configuration section header (e.g. [event_on_demand{user_logged_in}]).
If there is a Precondition in an event configuration header, there also must be a configuration for this event without any precondition. Is there for example an event configuration `event_on_demand{user_logged_in}`, there also has to be an event configuration `event_on_demand`! The event configuration with the precondition inherits all the parameters from the event configuration without precondition.

If the event is fired, first it will be checked which preconditions are true. If there is no precondition true, the configuration without precondition is used. Is one precondition true, the configuration is used, which is bound to this precondition. If more than one of the preconditions are true, the most specific event configuration is used (which is the configuration with the most matching options).

A small example for a better understanding:

While installing software it may be necessary to reboot the computer. Is there currently a user logged on, you should warn about the pending reboot. This warning should have a timeout and it may make sense to ask the user, if the reboot should be canceled (at the moment).

Is there no user logged on, it makes no sense to ask and wait for an answer. So in this case the reboot should take place immediately.

To handle these different situations, we configure the `event_on_demand` in the following way:

- We define a Precondition `user_logged_in` which comes true if a user is logged on to the system (`user_logged_in = true`).
- In the default configuration for the event `event_on_demand` (without any Precondition) we set `shutdown_warning_time = 0` (immediate reboot without warning).
- At the configuration `event_on_demand{user_logged_in}` we set `shutdown_warning_time = 300` (warning with 300 seconds timeout).

**Working Window**

To limit the functionality of an event to a certain time frame a so called working_window can be configured for all events.

The working_window feature requires `opsi-client-agent` >= Version 4.1.0.0-39.

To enable the working_window feature the key `working_window` has to be added to the configuration of the event. If this key is not present, has no value, or an invalid value it is treated as empty and the event has no time limit.

Starttime and endtime must be entered in the format hh:mm and separated by a hyphen. Whitespaces between starttime and endtime are not allowed!

All events support the working_window feature. Configuration is done by adding the host parameter `working_window` to the event of your choice. This can be achieved by using the `opsi-configed`, or via `opsi-admin` on the `opsi-config-server`.

The following example shows how to configure a working_window for the event `gui_startup` using `opsi-admin`.

See Section 6.1.3.12, “Configuration via web service (Host Parameter)” for how to add host parameter.
using \textit{opsi-configed}.

Example 1: Adding an empty \textit{working_window} for the event \textit{event\_gui\_startup} globally. The time restriction has to be configured client specific (see example 3).

\begin{verbatim}
opsi-admin -d method config\_createUnicode opsiclientd.event\_gui\_startup.working\_window
\end{verbatim}

Example 2: Adding a \textit{working_window} with a timeframe between 20:00 and 07:00 for the event \textit{event\_gui\_startup} globally.

\begin{verbatim}
opsi-admin -d method config\_createUnicode opsiclientd.event\_gui\_startup.working\_window
"gui\_startup.working\_window" "20:00-07:00"
\end{verbatim}

Example 3: Client specific configuration of the \textit{working_window} using a timeframe from 07:00 to 19:00 for the event \textit{event\_gui\_startup}.

\begin{verbatim}
opsi-admin -d method config\_State\_create opsiclientd.event\_gui\_startup.working\_window
"client.domain.de" "07:00-19:00"
\end{verbatim}

If the starttime is higher than the endtime the \textit{working_window} will be valid over the nightly day change (23:59-0:00).

Example during the day (starttime < endtime): working\_window=07:00-19:00
Example during the night (starttime > endtime): working\_window=20:00-07:00

For an exemplary "working\_window=20:00-07:00" the opsiclientd log would look like this:

\begin{verbatim}
[5] [<timestamp>] [ event processing gui\_startup] We have now: 2019-11-05 01:02:13.993000 (EventProcessing.pyo\|1121)
[5] [<timestamp>] [ event processing gui\_startup] Working Window configuration
starttime: 2019-11-04 20:00:00 endtime:
2019-11-05 07:00:00 systemtime now: 2019-11-05 01:02:13.993000
(EventProcessing.pyo\|1138)
[5] [<timestamp>] [ event processing gui\_startup] We are in the configured working
window (EventProcessing.pyo\|1139)
\end{verbatim}

\textbf{Proxysupport-Configuration}

In the global section of opsiclientd.conf you have the option to define a proxyserver that will be used by the opsi-client-agent. If a proxyserver is defined in config, all HTTP- and HTTPS-Connection of the opsiclientd will be redirected to this proxyserver.
# Use a proxy for connecting configservice

```bash
# proxy_mode:
# 'system' will try to check the system setting,
# 'static' to use proxyurl from configfile/hostparameter
# proxy_url usage: http://<user>:<password>@<proxy-url>:<proxy-port>
# Example: http://proxyuser:proxypass123@proxy.domain.local:8080
proxy_mode = static
proxy_url =
```

This proxy settings allows also to use a proxyserver, that require authentication. In that case you must define the credentials as shown in the configuration snippet.

⚠️ The option `proxy_mode` is reserved for the value `system` to use on the system proxy settings. This feature is not implemented yet. Therefore the only option that works at the moment is the `static` mode.

## Event configuration to control which products will be processed

With this new feature it's possible over the configuration to control the list of products, that will be processed in Events with product groups:

There are (basically) two ways to use this control:

### Blacklisting (excluding):

The option `exclude_product_group_ids` allows to configure a comma separated list of product Groups. The members of these groups will be excluded from the actual Event. Also if action request is set for this products. This products will be ignored in this event, but the action requests will not be changed.

### White listing (including):

The option `include_product_group_ids` allows to also configure a comma separated list of products Groups. The members of this groups are the only products, that will be processed from the actual event if they have set action requests.

You can use these options globally from the default-Event. From that point this settings will be used in every event. You can also set these options in a special event. If you use the option on `event_on_demand`, you can control which products will not be installed in push installations, although they have an action request. On normal restart of the client, the products will be installed from `gui_startup` (default event) at startup. CAUTION: For Clients that work in WAN/VPN-mode you must set this options in sync-event and also in the cacheservice-section, because the cache service have no access to the configuration of main sync-event.

⚠️ Product dependencies will not be observed by this feature. That means that you have to observe the process in order to prevent dependency issues.
Configuration via configuration file

On a 64bit Windows the configuration file is `c:\program files (x86)\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf`.

On a 32bit Windows the configuration file is `c:\program files\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf`.

This configuration file is UTF-8 encoded. Any changes using editors which do not support this encoding (e.g. notepad.exe) may destroy any umlaut in this file.

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

A sample `opsiclientd.conf`:

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

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     global settings                                                 =
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

[global]

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

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

# Client id.
host_id =

# Opsi host key.
opsi_host_key =

# Verify opsi server certs
verify_server_cert = false

# Verify opsi server certs by ca
verify_server_cert_by_ca = false
```
# On every daemon startup the user login gets blocked
# If the gui starts up and no events are being processed the login gets unblocked
# If no gui startup is noticed after <wait_for_gui_timeout> the login gets unblocked
# Set to 0 to wait forever
wait_for_gui_timeout = 120

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

# Use a proxy for connecting configservice
# proxy_mode:
# 'system' will try to check the system setting,
# 'static' to use proxyurl from configfile/hostparameter
# proxy_url usage: http://<user>:<password>@<proxy-url>:<proxy-port>
# Example: http://proxyuser:proxypass123@proxy.domain.local:8080
proxy_mode = static
proxy_url =

# Try to Suspend Bitlocker before rebooting the client
suspend_bitlocker_on_reboot = false

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

# Connection timeout.
connection_timeout = 30

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

# If this option is set, the local system time will be synced with time from service
sync_time_from_service = false

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;     depot server settings                                           -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[depot_server]
# Depot server id
depot_id =

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

drive =

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

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; - cache service settings
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[cache_service]
# Maximum product cache size in bytes
product_cache_max_size = 500000000
# Members of this ProductGroups will be excluded from processing
exclude_product_group_ids =
# Only members of this ProductGroups will be excluded from processing
include_product_group_ids =

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

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

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

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

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

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

# The maximum number of authentication failures before a client ip
# is blocked for an amount of time.
max_authentication_failures = 5

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; - notification server settings
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[notification_server]
# The network interfaces to bind to.
# This must be the IP address of an network interface.
# Use 0.0.0.0 to listen to all interfaces
interface = 127.0.0.1

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

# Port for popup notification server
popup_port = 45000

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

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

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

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

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     action processor settings                                       -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[action_processor]

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

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

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

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     events                                                          -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[event_default]
; === Event configuration
# Type of the event (string)
type = template
# Interval for timer events in seconds (int)
interval = -1
# Maximum number of event repetitions after which the event will be deactivated (int, -1 = forever)
max_repetitions = -1
# Time in seconds to wait before event becomes active (int, 0 to disable delay)
activation_delay = 0
# Time in seconds to wait before an event will be fired (int, 0 to disable delay)
notification_delay = 0
# Event notifier command (string)
event_notifier_command = %opsiclientd_notifier.command% -s notifier\event.ini
# The desktop on which the event notifier will be shown on (current/default/winlogon)
event_notifier_desktop = current
# Block login while event is been executed (bool)
block_login = false
# Lock workstation on event occurrence (bool)
lock_workstation = false
# Logoff the current logged in user on event occurrence (bool)
logoff_current_user = false
# Get config settings from service (bool)
get_config_from_service = true
# Store config settings in config file (bool)
update_config_file = true
# Transmit log file to opsi service after the event processing has finished (bool)
write_log_to_service = true
# Shutdown machine after action processing has finished (bool)
shutdown = false
# Reboot machine after action processing has finished (bool)
reboot = false
# Members of this ProductGroups will be excluded from processing
exclude_product_group_ids =
# Only members of this ProductGroups will be excluded from processing
include_product_group_ids =

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

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

; === Action processing
# Should action be processed by action processor (bool)
process_actions = true
# Type of action processing (default/login)
action_type = default
# Update the action processor from server before starting it (bool)
update_action_processor = true
# Command which should be executed before start of action processor
pre_action_processor_command =
# Action processor command (string)
action_processor_command = %action_processor.command%
# The desktop on which the action processor command will be started on (current/default/winlogon)
action_processor_desktop = current
# Action processor timeout in seconds (int)
action_processor_timeout = 10800
# Command which should be executed before after action processor has ended
post_action_processor_command =
[event Gui_startup]
super = default
type = gui startup
name = gui_startup
block_login = true

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

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

[event Gui_startup{installation_pending}]
name = gui_startup
active = true
[event_on_demand]
super = default
type = custom
name = on_demand

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

[event_software_on_demand]
super = default
type = sw on demand

[event_sync]
super = default
type = template
process_actions = false
event_notifier_command =
sync_config_to_server = true
sync_config_from_server = true
cache_products = true
cache_dynamic_bandwidth = true

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

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

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

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

[event_user_login]
super = default
type = user login
action_type = login
active = false
action_message = Starting to process user login actions.
block_login = false
process_shutdown_requests = false
get_config_from_service = false
update_config_file = false
write_log_to_service = false
update_action_processor = true
event_notifier_command = %opsiclientd_notifier.command% -s notifier\userlogin.ini
event_notifier_desktop = default
action_processor_command = %action_processor.command% /sessionid %service_session%
/allloginscripts /silent
action_processor_desktop = winlogon
action_processor_timeout = 300

[event_on_shutdown]
super = default
type = custom
name = on_shutdown
active = False

[event_on_shutdown{installation_pending}]
name = on_shutdown
active = False

[event_silent_install]
super = default
type = custom
name = silent_install
event_notifier_command = process_shutdown_requests = false
action_processor_productIds = swaudit,hwaudit
action_processor_command = %action_processor.command% /productlist
%action_processor_productIds% /silent
action_processor_desktop = winlogon
action_processor_timeout = 300

[event_timer_silentinstall]
super = silent_install
type = timer
active = false
interval = 21600

[precondition_user_logged_in]
user_logged_in = true

[precondition_cache_ready]
config_cached = true
products_cached = true

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

[precondition_installation_pending]
installation_pending = true

**Configuration via web service (Host Parameter)**

The opsiclientd configuration can be changed by the *host parameter* tab at the opsi management interface.

The entries in the *host parameter* have to be according to the following patterns:

`opsiclientd.<name of the section>.<name of the key>`

Example:

`opsiclientd.event_gui_startup.action_warning_time = 20`  
set in the configuration file `opsiclientd.conf` in the section `[event_gui_startup]` the value of `action_warning_time` to the value 20.

The following figure shows how to change the serverwide general configure via `opsi-configed`
Setting the server default `opsiclientd` configuration

Using the context menu you may choose `add property` to set a new key/value pair.

To delete a server default, please use the `opsi-admin` tool:

Example:

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

It is also possible to manipulate these entries client specific via `opsi-configed`.

To delete a client specific entry, please use the `opsi-admin` tool:

Example:

```
@opsi-admin> method configState_delete
"opsiclientd.event_gui_startup.action_warning_time" "myclient.uib.local"
```
Figure 62. client specific opsiclientd configuration via opsi-configed

Logging

The opsiclientd logs to:
C:\opsi.org\log\opsiclientd.log.

All log information will be transferred to the opsi-config-server via web service. At the server you find these log infos at /var/log/opsi/clientconnect/<ip-or-name-of-the-client>.log. They are presented in the opsi configed at the tab logfiles / client connect.

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

There are the following log levels:

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

Example:

(...)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config 'sync_completed(cache_ready)' added to event generator 'sync_completed' (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config 'gui_startup' added to event generator 'gui_startup' (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config
'gui_startup{cache_ready}' added to event generator 'gui_startup'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config 'on_demand' added
to event generator 'on_demand'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config
'sync_completed{cache_ready_user_logged_in}' added to event generator 'sync_completed'
(Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config
'gui_startup{user_logged_in}' added to event generator 'gui_startup'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config 'sync_completed'
added to event generator 'sync_completed'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config
'software_on_demand' added to event generator 'software_on_demand'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Event config
'on_demand{user_logged_in}' added to event generator 'on_demand'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] Updating config file:
'C:\Program Files (x86)\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf'
(Config.pyo|287)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] No need to write config file
'C:\Program Files (x86)\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf', config
file is up to date   (Config.pyo|318)
[5] [Mar 22 10:17:46] [ event processing gui_startup ] No product action requests set
(EventProcessing.pyo|591)
[5] [Mar 22 10:17:49] [ event processing gui_startup ] Writing log to service
(EventProcessing.pyo|247)
[6] [Mar 22 10:17:49] [ opsiclientd ] shutdownRequested: 0
(Windows.pyo|340)
[6] [Mar 22 10:17:49] [ opsiclientd ] rebootRequested: 0
(Windows.pyo|326)
[5] [Mar 22 10:17:49] [ opsiclientd ] Block login now set to 'False'
(Opsiclientd.pyo|111)
app (pid 1620)   (Opsiclientd.pyo|148)
[6] [Mar 22 10:17:49] [ event processing gui_startup ] Stopping notification server
(EventProcessing.pyo|225)
[6] [Mar 22 10:17:51] [ control server ] client connection lost
(Message.pyo|464)
[6] [Mar 22 10:17:52] [ event processing gui_startup ] Notification server stopped
(Message.pyo|651)
[5] [Mar 22 10:17:52] [ event processing gui_startup ] =============
EventProcessingThread for event 'gui_startup' ended =============
(EventProcessing.pyo|1172)
[5] [Mar 22 10:17:52] [ opsiclientd ] Done processing event
'<ocdlib.Events.GUIStartupEvent object at 0x023CE330>'   (Opsiclientd.pyo|405)
[5] [Mar 22 10:19:41] [ opsiclientd ] Session
'HSzMB1wt0iBS6vH17m3roSr63TanFu' from ip '127.0.0.1', application 'opsi jsonrpc module
version 4.0.1' expired after 120 seconds   (Session.pyo|184)
[6] [Mar 22 10:19:41] [ opsiclientd ] Session timer <_Timer(Thread-20,
started daemon 2636)> canceled   (Session.pyo|120)
The opsi-login-blocker logging to the log file: C:\opsi.org\log\opsi_loginblocker.log.

opsiclientd infopage

According to the fact that there are a lot of subcomponents of the opsiclientd which work and log at the same time, the log file of the opsiclientd becomes complex.

In order to make it easier to understand how the different subcomponents work together, the opsiclientd has an own info page which visualizes the running tasks on a timeline. You may view this info page at the browser calling the url: https://<address-of-the-client>:4441/info.html
opsiclientd Bitlocker Suspend Feature

Clients with activated Bitlocker encryption, with manual password entry at boot, prevent the unattended installation of software and patches.

Just like with opsi-winst, it is now also possible for reboots triggered by events of opsiclientd to suppress the password input when booting.

This feature is inevitably associated with a security issue. During this process, the password is written to the hard disk as plain text and is therefore a potential weak point.

This feature is deactivated by default, in order to activate this option only on selected clients, a standard configuration must be created first:
The default value false corresponds to the value in the supplied `opsiclientd.conf`.

To set the host parameter via `opsi-admin`, the following command has to be executed on the `opsi-config-server` (in the example for a client with the opsi-host-Id `myclient.domain.de`):

```
opsi-admin -d method configState_create clientconfig.suspend_bitlocker_on_reboot myclient.domain.de true
```

This option can also be activated on clients that have not activated Bitlocker encryption and this option should not interfere with the operation of the `opsiclientd`.

**opsi-client-agent remote control**

The `opsiclientd` has its own web service interface which can be used to transmit commands to the `opsiclientd`. The possible commands can be divided in the following categories:

- send Messages (Popup)
- *Push* installation (start the event *on_demand*)
- other maintenance tasks

This can be done on the command line using the tool `opsi-admin` by calling one of the `hostControlSafe_*` methods. Calling one of these methods takes the parameter *hostid* which:

- can be `[*]` to send the command to all clients
- can be the name of a client (e.g., "pcbon4.uib.local")
- can be a list of client names according to the pattern `[<client1>, <client2>]`  
  e.g., ["pcbon1.uib.local", "pcbon2.uib.local"]
- may contain wildcards like *
  e.g., "pcbon4.*" or "pcbon*"

If a client isn't reachable (e.g. powerd off) you will get a message.

**Sending popup messages**

Using the `opsi-configed` you may send messages to the clients. Section 4.8.4, “Sending messages (Show popup message)”

At the command line you may do this with the tool `opsi-admin`:
opsi-admin -d method hostControlSafe_showPopup message *hostid

Example:

opsi-admin -d method hostControlSafe_showPopup "This is my message" "myclient.uib.local"

Push installations: start the event on demand

The *opsi-server* may send a command to the client that the client should process the configured action requests immediately. This is done by activating the event *on_demand* at the client.

This is possible using the *opsi-configed* and is described in chapter: Push installationen: start the event on demand

From the *opsi-server* the client can be instructed to execute the *product actions*.

Executing Events can also be done from the *opsi-configed*. Section 4.8.3, “Fire opsiclientd event (Push Installation)”

On the command line you may use *opsi-admin* to fire an event:

opsi-admin -d method hostControlSafe_fireEvent event *hostIds

Example:

opsi-admin -d method hostControlSafe_fireEvent "on_demand" "myclient.uib.local"

Additional maintenance tasks (shutdown, reboot,…..)

Using the control server port you may remote control the *opsiclientd*. In order to do this you have to authenticate yourself at the web service. This could be done either with the local administrator account (with a not empty password) or with the *opsi-host-Id* (FQDN, client name and DNS Domain name) as user name and the opsi-hostkey as password.

Using the *opsi-configed* you may choose the menu *opsiClient* or the context menu in the *Clients* Tab.
At the command line you also can initiate a client:

**shutdown:**

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

**reboot:**

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

### 6.1.4. Adapting the opsi-client-agent to your Corporate Identity (CI)

Adapting the *opsi-client-agent* to your Corporate Identity can be important for the user acceptance when rolling out opsi. By adding your corporate logo to the opsi background image, the users feel more familiar with the opsi installation instead of being puzzled by something unknown.

Since opsi-client-agent version 4.0.7.16 all graphic components of the opsi-client-agent (notifier, opsi-script, kiosk-client) are base on the same graphic libraries and may be customized in the same way. Colors can be configured in three different ways: as symbolic name (clRed), as hexadecimal value ($FF00FF) and as rgb value ((255,0,0)). There is a helper program set allows you simple to choose your colors and get the correct way to write the colors to the configuration file. The program is *opsi color chooser*.
As background graphic formats you may use a large number of different formats like: .bmp, .png, jpeg and so on. But all these formats are include a number of subformats. So for example is one png file displayed without any problem while an other, different png-file may not displayed in a correct way. There may also be a difference between the operating system platforms (e.g between Windows and Linux).

There is a helper program set allows you simply to check if a given bitmap file will be displayed correct or not: opsi bitmap viewer.

**Elements to be patched: opsi-winst**

The files to be configured for opsi-winst are to be found in the directory `/var/lib/opsi/depot/opsi-client-agent/files/opsi/opsi-winst/winstskin`:

- `bg.png`
  This is the *opsi-winst* background image, where during installation text messages and product logos are shown.

- `skin.ini`
  This is the configuration file to specify the position, font and color of text messages during installation.

**Elements to be configured: opsiclientd**

In the directory `/var/lib/opsi/depot/opsi-client-agent/files/opsi/dist/notifier` are the files to configure the look of the notifiers. Each notifier has an image and a configuration file:

- `block_login.bmp`
  background image of the login blocker notifier.

- `block_login.ini`
  configuration file of the login blocker notifier.

- `event.bmp`
  background image of the server connection event notifier.

- `event.ini`
  configuration file of the server connection event notifier.

- `action.bmp`
  background image of the action notifier (software installation).

- `action.ini`
  configuration file of the action notifier.

- `shutdown.bmp`
  background image of the shutdown/reboot action notifier.

- `shutdown.ini`
  configuration file of the shutdown/reboot action notifier.

- `popup.bmp`
  background image of the popup message notifier.
• **popup.ini**  
  configuration file of the popup message notifier.

• **userlogin.bmp**  
  background image of the user login event notifier.

• **userlogin.ini**  
  configuration file of the user login event notifier.

**Elements to be configured: kioskclient**

This chapter is only relevant for opsi-client-agent versions <= 4.1.0.0. Since 4.1.1.0 the kiosk is outsourced and available as opsi-client-kiosk package. For description of the opsi-client-kiosk see: [Section 9.17, “opsi Software On Demand - opsi-client-kiosk (free)”](#)

The Headers list from the Main window (1) is customizable to the desire of the client. To that, there are two files which play a roll:

The header of the main windows (1) is customizable.

There are three files which are relevant:

• **header.png**
• **logo.png**
• **opsiclientkiosk.ini**

The **opsiclientkiosk.png** holds the picture which will be loaded in this area.

The **opsiclientkiosk.ini** defines the text and its representation which will be shown in this area.

Example:
You will find templates for these files under `/var/lib/opsi/depot/opsi-client-agent/files/opsiclientkiosk/opsiclientkioskskin` or `C:\Program Files(x86)\opsi.org\opsi-client-agent\opsiclientkiosk\opsiclientkioskskin`

**Protect your CI changes from updates: the custom directory**

(available since opsi-client-agent version 4.0.2.3)

The custom directory can be used to protect your configuration changes during opsi-client-agent updates: (`/var/lib/opsi/depot/opsi-client-agent/files/opsi/custom`). During server updates of opsi-client-agent the whole custom directory will be saved and restored after the update, so that your custom changes will persist.

- **custom/config.ini**
  Values from this config file override values from the default `cfg/config.ini`. Except of the values for `pckey` and `bootmode`, which never are picked from that file. Add to your custom config file only those values, that are different from the default settings.

- **custom/winstskin/*.***
  All the files from this directory will be copied to the clients `C:\Program Files (x86)\opsi.org\opsi-client-agent\custom\winstskin` directory during installation of the opsi-client-agent on the client. This `winstskin` directory, if it exists, since opsi-winst Version 4.11.3.4. is the preferred one. It must
contain all required winstskin files and configurations, for the content of the default directory is ignored.

- **custom/notifier/*.***
  All the files from this directory will be copied to the clients C:\Program Files (x86)\opsi.org\opsi-client-agent\notifier directory during installation of the opsi-client-agent and overwrite the files from the server side files/opsi/dist/notifier/ directory.

- **custom/opsiclientd.conf**
  If it exists, the custom/opsiclientd.conf will be copied to the clients C:\Program Files (x86)\opsi.org\opsi-client-agent\opsiclientd directory during installation of the opsi-client-agent and overwrites the default opsiclientd.conf from the server side files/opsi/dist/opsiclientd/ directory. So the custom opsiclientd.conf must contain all the required configuration entries.
  
  **Attention:**
  Using a custom opsiclientd.conf is not recommended. To customize your client configuration, use the host parameter configuration for single features as described in the opsi-client-agent chapter. Using a custom opsiclientd.conf is applicable for very complex configurations only. By using a custom opsiclientd.conf, after each update of opsi-client-agent it is required to check the server default file files/opsi/dist/opsiclientd/opsiclientd.conf for changes to be patched to your custom opsiclientd.conf.
  
  So: hands off this feature, unless you really know what you are doing!

- **custom/opsiclientkioskskin/*.***
  All the files in these directory will be copied, by the installation of an opsi-client-agent to C:\Program Files(x86)\opsi.org\opsi-client-agent\custom\opsiclientkioskskin. If available, this directory (opsiclientkioskskin) will have the preference over others.

To avoid errors by the change of rights, the following helps:

```
opsi-setup --set-rights /var/lib/opsi/depot/opsi-client-agent
```

### 6.1.5. Blocking the user login with the opsi-Loginblocker

To prevent a user login before all installations are completed, opsi provides the optional *opsi-login-blocker*.

**opsi loginblocker at Windows 2000 to XP (NT 5)**

The *opsi-login-blocker* is implemented as the Gina opsigina.dll. Gina means *Graphical Identification and Authentication* and is the official Microsoft hook to manipulate the login process.

If you already have a special Gina-DLL installed, which is different from the original Microsoft msgina.dll (e.g. Novell nwgina.dll), you should not install the opsi-login-blocker without consulting uib or [https://forum.opsi.org](https://forum.opsi.org). 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 `opsi-login-blocker` is installed or not is configured by the switch `LoginBlockerStart=on/off` in section `[opsi-client-agent-install]` of the client configuration.

### opsi loginblocker at NT 6 (Win 7 & Co)

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

#### 6.1.6. Subsequent installation of the opsi-client-agents

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

### Installation of the opsi-client-agent from a master image or as exe

In order to install the opsi-client-agent from a prepared (sysprep) master image, the opsi-client-agent has to be (re)installed while the clone awakes and get a new personality.

To do this use the following steps:

- Copy from the share `opsi_depot` the complete content of the directory `opsi-client-agent` in a temporary directory on the master.
- Edit there the file `files\opsi\cfg\config.ini`:
  - In section `[installation]` set for `service_user=` the login name of a user that is member of the opsiadmin group.
  - NOT RECOMMENDED: In section `[installation]` set for `service_password=` the uncoded password of this user. Better:
  - In section `[installation]` set for `service_hidden_password=` the base64 encoded password of this user. For encoding of the password you may use the opsi-winst function `base64EncodeStr(<string>)` or an online service like `http://www.base64encode.org/`.
  - If `service_hidden_password=` has any value the key `service_password=` will be ignored.
  - In section `[opsiclientd]` set for `config_service.url` = the web service address of your opsi-config-server (e.g. `https://192.168.1.10:4447/rpc`)
- Make sure that the script `silent_setup.cmd` from the temporary directory is called after the clone has its new personality.
- After the call of `silent_setup.cmd` is finished, the temporary directory should be deleted.

If you like you may pack the temporary directory to a self extracting exe with final program start using a tool like filzip.

### 6.1.7. The Systray Program of the opsi-client-agent

The systray program of the `opsi-client-agent` focuses on the following targets:

- Notifying the user in regular (and configurable) Intervals on pending Installations. (Optional)
• Notifying the user on pending Installations on demand by using the context menu.
• Possibility for the user to start the installations.

![Figure 65. Message window of the opsi-systray program](image)

Figure 65. Message window of the opsi-systray program

![Figure 66. Context menu (right mouse click) of the opsi-systray program](image)

Figure 66. Context menu (right mouse click) of the opsi-systray program

### Controlling the opsi systray program via the `opsi-client-agent` product properties:

- **systray_install**
  - (true / false) Install the opsi systray program?
  - Default = false

- **systray_check_interval**
  - Interval in minutes to check for pending action requests.
  - Default = 180 (Small values here give heavy load to the server)
  - The value 0 means: no checks at all..

- **systray_request_notify_format**
  - Format of action request notification.
  - Possible Values:
    - "productid : request",
    - "productname : request",
    - "productname productversion : request"
  - Default: "productname : request"

### Logs of the opsi systray program:

The program logs to `%Appdata%\opsi.org\log`. That is the `opsi.org\log` directory in the Appdata directory of the loggedin user.

For Example:

`C:\Users\<username>\AppData\Roaming\opsi.org\log\`

See also Chapter *opsi Software On Demand (Kiosk-Mode): Section 9.17, “opsi Software On Demand - opsi-client-kiosk (free)”*

### 6.2. Registry Entries
6.2.1. Registry entries of opsiclientd

opsi.org/general

- bootmode= <bkstd | reins>
  Stores whether the computer is newly installed or not.

opsi.org/shareinfo

- depoturl
  <URL that refers to the location of the software packages. Pattern: protocol:\\server\share\dir>
  Example:
  smb:\\opsi-server\\opsi depot

- depotdrive
  <Drive letter on which depoturl is mounted>
  Example: P: (including the colon)

6.2.2. Registry entries of opsi-winst

opsi.org/winst

These registry entries are controlled by opsi-winst and should not be changed.

```
"LastLogFilename"="C:\\TMP\\syslogin.log"
"ContinueLogFile"=dword:00000000
"RebootRequested"=dword:00000000
"SendLogToService"=dword:00000001
"NumberOfErrors"=dword:00000000
"ShutdownRequested"=dword:00000000
```

6.3. opsi-client-agent web service

6.3.1. Reading log files

- log_read
  Reads an opsi log file available on the client.
  Parameters: logType *extension *maxSize
  Possible logType values are opsiclientd, opsiclientdguard, opsi-client-agent, opsi-script, opsi_loginblocker, notifier_block_login and notifier_event.
  Rotated log files (log.1, log.2, _1.log, _2.log etc.) can be selected with the parameter extension.
  Possible values are 0-9.
  The maxSize parameter limits the output to the specified value in bytes.
7. Security

7.1. Introduction

Opsi is a powerful tool for the administration of many clients.

According to that fact, the opsi-server has to be in the focus of security considerations.

If you control the opsi-server, you are in control of all the clients, that are connecting to that opsi-server.

How much time and money you should spend for hardening your opsi-server, depends on your needs regarding security and the operational environment for using opsi. So for example an opsi-server in the cloud is more endangered than an opsi-server in a secured network.

In the following chapter we have collected the most important issues and problems.

At this point we say thank you to all customers and users which informed us about security problems and helped us to improve the security of the opsi system. If you find any security problem, please inform us (info@uib.de) before disclosing the security vulnerability in public.

7.2. Stay tuned

Information about security relevant updates and tasks are published at the news area at the opsi forum:

7.3. General server security

The opsi software cannot be more secure than the underlying operating system. So please make sure to update your server with the security updates of your Linux distribution. This has to be done not only for the opsi-config-server, but also for all the opsi-depot-server.

It may help you to install programs which inform you by email if there are new updates available.

Debian, Ubuntu
   apticron

RHEL, CentOS
   yum-updatesd

There are a lot of possibilities to enhance the security of your Linux server. But this is not the task of this manual.

We would be happy to help you with this task as part of a support contract.
7.4. Client authentication at the server

The client authenticates itself using the FQDN as username and the `opsi-host-key` as password.

The `opsi-host-key` is stored at the client in the file:

```
%programfiles%\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf
```

which is readable with administrative privileges only.

The `opsi-host-key` is stored at the server in the used backend (e.g. at `/etc/opsi/pckeys`).

In addition to this authentication, you may tell the `opsiconfd` to check if the client IP address matches the given FQDN. To activate this check, set at the `/etc/opsi/opsiconfd.conf`:

```
verify ip = yes
```

and restart `opsiconfd`:

```
systemctl restart opsiconfd.service
```

Do not use this feature if you are not really sure, that your name resolution works properly in both directions for all clients.

7.5. Server authentication at the client

Since opsi 4.0.1 there are different possibilities to check the trustworthiness of the contacted server.

**Do not use them in combination. Choose only one way or you will be locked out from your client.**

7.5.1. Variant 1: `verify_server_cert`

At the first contact to a opsi-server, the client will accept the given SSL certificate and store it at `C:\opsi.org\opsiclientd\server-certs`.

On any subsequent contact, the client creates a random string and uses the *public key* of the stored certificate to encrypt this string (and the own access parameters). These encrypted data will be sent to the server.

The server uses the *private key* of its own SSL certificate to decrypt the data and sends the decrypted random string back to the client.

Now the client checks if the correct string was sent back. If not, the communication to the server will be aborted.

You can prevent this way that somebody directs your clients to a wrong server, e.g. by manipulating the DNS. If you setup a new server, you may migrate the SSL certificate from the old to the new server without problems. And you must not deploy any certification authority (CA).
The disadvantage of this method is, that a *man-in-the-middle* attack is still possible.

This security method checks the communication between client and *opsi-config-server*.

Using the opsi WAN extension and as *clientconfig.depot.protocol webdav*, also the communication to the *opsi-depot-server* is checked.

Section 9.11.3.1, “Communication Protocol for accessing an opsi-depot”

To activate this check, set at the *opsiclientd.conf* in the section *[global] the option:

```
verify_server_cert = true
```

Run the following command at your *opsi-config-server* to create this configuration entry for all clients:

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

Now you can activate this using the *opsi-configed* at the *Server configuration* or at the *Host parameter* of selected clients by changing the value from *false* to *true*.

Be very careful with activating "verify_server_cert", for in case of improper configuration your clients will refuse the connection!

7.5.2. Variant 2: verify_server_cert_by_ca

This variant works just like SSL certificates are checked in your browser. The given SSL certificate will be accepted, if it is issued for the exact FQDN (*commonName*) of the server (or if the DNS verifies that this is the FQDN matching the IP address of the server) and the certificate is issued and signed by the *uib gmbh*.

Is one of these conditions not true, the communication to the server will be aborted.

This method is more secure than the first one. But you will have to buy the certificates from *uib gmbh*. For prizes and conditions have a look at the prize list of *uib gmbh*:

http://uib.de/en/opsi_support/index.html

Any profits from selling these certificates will be invested in the maintenance of the opsi security.

To activate this security method, set at the *opsiclientd.conf* in the section *[global] the option:

```
verify_server_cert_by_ca = true
```

Run the following command at your *opsi-config-server* to create this configuration entry for all clients:
Now you can activate this using the `opsi-configed` at the `Server configuration` or at the `Host parameter` of selected clients by changing the value from `false` to `true`.

Be very careful with activating "verify_server_cert_by_ca", for in case of improper configuration your clients will refuse the connection!

### 7.6. Authentication at the control server of the client

The `opsiclientd` provides a web service interface, which allows remote control of the `opsiclientd` and thus remote control of the client.

(Section 6.1.3.16, “opsi-client-agent remote control”).

In order to access this interface authentication is required. You may authenticate as a local administrator with a not empty password, or with an empty user name and the `opsi-host-key` as password.

### 7.7. Admin network configuration

The idea of an `admin network` is to ban any administrative access from the standard production network and allow these accesses only from a special `admin network`.

With opsi all `opsi-clients` need restricted access to the `opsi web service`, which allows them to read and change their own data. Administrative access with further privileges is granted to members of the unix group `opsiadmin` only.

If you configure an `admin networks` parameter, all administrative accesses are restricted to these network(s).

Setting the option `[global] admin networks` at the `/etc/opsi/opsiconfd.conf` will restrict the administrative access to the `opsiconfd` to connections coming from the specified network address(es). You may give multiple addresses separated by comma.

Non administrative access may also come from other networks.

The default is:

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

and allows administrative access from all networks.

A configuration like e.g.
admin networks = 127.0.0.1/32, 10.1.1.0/24

restricts administrative access to the server itself and to the network 10.1.1.0/24.

7.8. The user pcpatch

With opsi 4 the user pcpatch is used just by the opsi-client-agent to mount the depot share (opsi_depot).

Exceptions are the products:

- opsi-wim-capture and opsi-local-image-capture which use pcpatch to mount the share opsi_depot_rw
- opsi-clonezilla which use pcpatch to mount the share opsi_images

The password of the user pcpatch is usually stored and transmitted encrypted. Under special circumstances it might be possible to catch the clear password. To reduce risks arising from that, you should do the following:

Deny for the user pcpatch the access to all other shares than the opsi_depot share. You should do this by adding the following entry to all share definitions (besides the opsi_depot) at the /etc/samba/smb.conf:

```
invalid users = root pcpatch
```

Alternative

At the /etc/samba/smb.conf restrict privileges for the user pcpatch to global read only by setting in the [global] section:

```
read list = pcpatch
```

For the products opsi-wim-capture and opsi-local-image-capture the share opsi_depot_rw must have write permission for pcpatch. For the product opsi-clonezilla the share opsi_images must have write permission for pcpatch.

As an additional task you should frequently change the password of the user pcpatch. You may set the password to a random string which no one knows (besides opsi). You may do this by calling the following command e.g by a cronjob:

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

If you are not using netboot products that require the possibility to login as user pcpatch you can disable the login for that user. To do so please change the shell of the user pcpatch to /bin/false in the
file /etc/passwd. Since opsi 4.1 the default shell for the user `pcpatch` is `/bin/false`. You only need to take action if the system was set up using an earlier version.

### 7.9. Webservice access limitations

The file `/etc/opsi/backendManager/acl.conf` can be used to limit the access to specified methods and attributes of the returned values.

The limitation affects the base methods of the webservice. For those a restriction of users or groups and allowed attributes can be established.

The access should be limited to the used methods. If it is not clear what methods are being used one can refer to the output of opsiconfd about the accessed methods. This is logged to `/var/log/opsi/opsiconfd/opsiconfd.log` in case of a stop or restart.

More information about the webservice can be found at Section 5.4.1, “object oriented methods”.

### 7.10. Change the bootimage root password

The root password of the opsi linux bootimage is `linux123` by default. You may like to change this for security reasons. How to do this is described here: Section 8.2.1, “Parameters for the opsi linux boot image”
8. opsi products

8.1. Localboot products: automatic software distribution with opsi

Localboot products are all products that are installed by the opsi-client-agent after the computer started the installed OS. This is in contrast to the netboot products described below Section 8.2, “Netboot products”.

8.1.1. opsi standard products

The following localboot products are part of the default installation of opsi.

opsi-client-agent

The opsi-client-agent is the client agent of opsi and is described in detail above: see chapter Section 6.1, “opsi-client-agent”.

opsi-winst

The product opsi-winst is a special case. It contains the current opsi-winst. This does not have to be set to setup to update. Rather, part of the opsi-client-agent checks each time it is started whether a different version of the opsi-winst is available and fetches it if this is the case.

javavm: Java Runtime Environment

The product javavm provides the Java Runtime Environment for the clients, which is required for opsi-configed.

opsi-configed

The opsi graphical management interface packaged as application For Windows and Linux. See also chapter: Chapter 4, opsi-Management GUI: opsi-configed

jedit

Java based editor with syntax highlighting for opsi-winst scripts.

swaudit + hwaudit: Products for hard- and software-audit

The hwaudit and swaudit products are used for hardware and software inventory. The hardware data is collected via WMI and reported back to the server via the opsi web service. For the software inventory the information is taken from the registry key (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall)and reported back to the server via the opsi web service.
opsi-template

Template for creating your own opsi scripts. You can extract the template with:

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

or rename it at the same time:

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

See also the opsi-getting-started manual.

opsi-template-with-admin

Template for creating your own opsi scripts. You can extract the template with:

```
opsi-package-manager -x opsi-template-with-admin_<version>.opsi
```

or rename it at the same time:

```
opsi-package-manager -x opsi-template-with-admin_<version>.opsi --new-product-id myprod
```

See also chapter: "opsi WIM Capture"

shutdownwanted

Shuts down the computer when there are no further actions pending.

opsi-script-test

Large collection of opsi-script self tests. This can be used as a sample collection for working calls of opsi-script commands.

opsi-wim-capture

See also chapter: Section 9.4, “opsi WIM Capture”

opsi-winpe

Product for easy generation of an opsi-winpe See also the opsi-getting-started manual, chapter Creating a PE.
**opsi-uefi-netboot**

See also chapter: Section 9.6, “opsi with UEFI / GPT”

**opsi-set-win-uac**

Sets the UAC level via opsi.

**opsi-setup-detector**

See also chapter: Section 9.21, “opsi Setup Detector (free)”

**opsi-logviewer**

Text viewer with selection for log levels and events.
For Windows and Linux.

- The opsi-logviewer tool created by uib now also opens files that are compressed in the archive formats zip or gzip. This means that log files that are sent as an archive can be viewed directly. (If an archive contains several files, the contents are of all files are appended.)
- The setup script has been expanded to include Linux support so that opsi-logviewer can also be installed automatically on a Linux client.
- The new before-dependency on javavm ensures the this program can be started (because javavm copies the javaw.exe into the system directory)

**config-win10**

Configures various Windows 10 settings such as lock screen, hibernation boot, sending telemetry and update behavior.

Version 4.0.7 added new options.

- **change_power_plan** changes the power management profile.
- **config_updates** allows you to change the source of the updates. The updates are then downloaded either directly from Microsoft servers, a local peer-to-peer network or a peer-to-peer network from the Internet. The disable option is meanwhile moved into a separate property called disable_updates.
- **defer_upgrade** postpones updates and upgrades. Updates can be postponed by four weeks and upgrades by eight months. It should be noted that security-relevant updates are installed despite the defer option. However, feature updates are not installed.
- **disable_advertising_id** deactivates the so-called Advertising ID. This stores data about the browser history in order to display user-specific advertising.
- **disable_cortana** deactivates the Cortana voice assistant. This collects various data about input and transfers this data to Microsoft servers.
- **disable_customer_experience** disables collecting data related to application usage data.
• **disable_defender** disables the anti-virus protection included with Windows 10 called Defender.

• **disable_fast_boot** deactivates fast boot and ensures that the standard opsi-event gui_startup works properly.

• **disable_font_streaming** ensures that fonts not installed on the system are streamed from the Internet.

• **disable_handwrite_sharing** A special feature is the use of Windows 10 on tablet PCs. Here, data about handwriting is collected and sent to a Microsoft server.

• **disable_location_sensors** disables collecting data about the current geolocation of the device.

• **disable_lock_screen** disables the lock screen.

• **disable_mac** deactivates a service that collects data about currently logged in users and transfers it to Microsoft.

• **disable_mrt** deactivates the use of the Malware Removal Tool, MRT for short. This service scans existing files on the computer's hard drive at regular intervals and compares them with a list of potentially dangerous software.

• **disable_onedrive_sync** disables the OneDrive file synchronization.

• **disable_sending_feedback** makes it possible to influence the transfer of data to Microsoft in the event of application crashes.

• **disable_smbv1** disables the SMB v1 protocol.

• **disable_telemetry** makes it possible to limit the amount of data collected. A lot of data is transferred as standard. If the property is set to true, Windows is set so that only security-relevant data is transferred. This is the lowest level. This security level can only be set in the Windows 10 Enterprise and LTSB version. In the other versions of Windows 10, the next lowest level is applied, Basic.

• **disable_updates** blocks connections to Microsoft update sources when set to true. Setting the property to false enables these connections again.

• **disable_wifi_sense** deactivates the service called Wifi Sense. This service enables saved WLAN configurations to be shared with contacts.

• **flashplayer_autorun** There is a security vulnerability in Windows 10 with the Adobe Flashplayer. It is recommended to deactivate the autorun feature of the flash player. With false the Flashplayer is no longer started.

• **online_search** Online results are also provided for every search using the integrated search bar in the taskbar. true enables such an online search, false disables it.

• **sync_settings** If you use Windows 10 in combination with a Microsoft account, it is possible to synchronize your settings with the current Microsoft account. If you set the property sync_settings to false this will be deactivated.

• **disable_automatic_logon_on_reboot**: When true it deactivates an automativ Winlogon on automatic reboot.

• **local_wsus_available**: Only affects Windows Updates: When true a connection to a local WSUS server is possible.
- **oobedisableprivacyexperience**: Only affects Windows 10 1809 and newer. Deactivates OOBE DisablePrivacyExperience, if true.

- **no_new_app_install_notification**: If this is set to true this property deactivates notifications on app updates.

```plaintext
[ProductProperty]
type: bool
name: disable_fast_boot
description: Disable Fastboot for proper opsi startup
default: True

[ProductProperty]
type: bool
name: disable_lock_screen
default: True

[ProductProperty]
type: bool
name: disable_telemetry
description: Disable telemetry data transmission
default: True

[ProductProperty]
type: bool
name: disable_cortana
description: Disable Cortana assistant
default: True

[ProductProperty]
type: bool
name: disable_customer_experience
description: Disable customer experience program
default: True

[ProductProperty]
type: bool
name: disable_mrt
description: Disable Malicious Software Removal Tool
default: True

[ProductProperty]
type: unicode
name: config_updates
multivalue: False
editable: False
description: Set Windows-Update behavior
values: ["AllowPeerToPeer", "LocalPeerToPeer", "MicrosoftOnly"]
```
default: ["MicrosoftOnly"]

[ProductProperty]
type: bool
name: disable_mac
description: Disable Microsoft Account communication
default: False

[ProductProperty]
type: bool
name: disable_advertising_id
description: Disable Microsoft Advertising ID
default: False

[ProductProperty]
type: bool
name: disable_updates
description: Disable Windows Updates
default: False

[ProductProperty]
type: bool
name: disable_defender
description: Disable Microsoft Windows Defender
default: False

[ProductProperty]
type: bool
name: disable_wifi_sense
description: Disable Wi-Fi Sense
default: False

[ProductProperty]
type: bool
name: disable_sending_feedback
description: Disable sending feedback and diagnostics
default: False

[ProductProperty]
type: bool
name: disable_font_streaming
description: Disable font streaming of not installed fonts
default: False

[ProductProperty]
type: bool
name: defer_upgrade
description: Defer Windows 10 Upgrade
default: True
config-winbase

Package for customizing the basic settings of the user interface, Explorer, etc.

opsi-auto-update

opsi-auto-update is a product to simplify the maintenance of the clients.

opsi-auto-update is not for clients running the WAN-extension

In essence, this product can be used to ensure that the installed products are up to date. The product sets all installed products, whose version is not identical to that on the server, for the client to setup.

Properties for managing exceptions:

- name: `products_to_exclude`
• description: (Blacklist) Which opsi product(s) should be excluded from update?
  List of products that should not be installed even if there is an update (such as windomain)

• name: products_to_exclude_by_regex
  • description: (Blacklist) Which opsi product(s) should be excluded from update (by regular expressions)?
  List of regular expressions that fit products that should not be installed even if there is an update (such as windomain)

• name: products_to_include
  • description: (Whitelist) Which opsi product(s) should be checked for update? ; If empty = all products
  Here you can enter a list of products to which the update check should be limited. Products that are not in this list are also not considered when checking for updates. Exception: If the list is empty, all products are checked.

Properties for managing includes:

• name: products_to_run_always
  • description: Which opsi product(s) should be installed via every update? (List will not be cleared after run)
  List of products which are set to setup every time opsi-auto-update is run.

• name: failed_products_to_setup
  • description: if true this also sets all failed products to setup on all clients
  If this is True, all products currently on failed are set to setup.

Properties for sequence control:

• name: shutdown_on_finish
  • description: if true we have a final shutdown if false we have no reboot / shutdown default: False
  Should a shutdown me made after the product has finished?

Special properties for local-image / vhd-reset:
See also: Section 9.9, “opsi vhd reset”
See also: Section 9.8, “opsi local image”

• name: do_cleanup
  • description: If false: skip restore before update
  This property is ignored if it is not a vhd or local image installation.
  For a vhd installation, do_cleanup=true executes opsi-vhd-control before the updates, thereby discarding all changes and restoring the saved state.
  For a local-image installation, do_cleanup=true executes opsi-local-image-restore before the updates, thereby discarding all changes and restoring the saved state.
  In both cases, information about action requests is also discarded. In order to be able to add or
remove products during a run of **opsi-auto-update**, there are the following two properties.

- name: *products_to_install*

  - description: Which opsi product(s) should be installed via update? (List will be cleared after run)
    List of products which are set to setup during the opsi-auto-update run. If the products have been installed successfully, they will be removed from this list.

- name: *products_to_uninstall*

  - description: Which opsi product(s) should be uninstalled via update? (List will be cleared after run)
    List of products which are set to uninstall during the opsi-auto-update run. If the products have been uninstalled successfully, they will be removed from this list.

- name: *do_merge*

  - description: If false: skip backup after update
    This property is ignored if it is not a vhd or local image installation.
    For a vhd installation, do_cleanup=true executes **opsi-vhd-control** with upgrade=true after the updates and thus all changes are saved.
    For a local-image installation, do_cleanup=true executes **opsi-local-image-backup** after the updates and thus all changes are saved.

Properties for debugging:

- name: *disabled*

  This property is for debugging purposes.
  If *true*, the product does not execute any actions.
  Default = *false*

- name: *rebootflag*

  Please do not change during the run. This should be *0* before starting.

- name: *stop_after_step*

  This property is for debugging purposes.
  If not *0* then this is the number of reboots after which to stop. Default = *0*

The **opsi-auto-update** product has a very low priority (-97), which is even less than that of opsi-vhd-control.

The **opsi-auto-update** product can be combined well with a cron job that executes the script

```
wake_clients_for_setup.py
```


For details see here: Section 5.10.1, “The cronjobs wake_clients_for_setup.py, opsi-auto-update and working_window”

### 8.1.2. Manipulating the installation sequence by priorities and dependencies

Since opsi 4.0, the installation order is determined by the opsi-server taking into account product
dependencies and product priorities.

- **Product dependencies**
  Defines dependencies and the necessary installation order between opsi-packages. A typical example is the dependency of Java programs on the Java Runtime Environment (javavm).

- **Product priorities**
  Priorities are used to push certain packages forward or backward in the order of installation. It makes sense to install service packs and patches first and a software inventory at the end of an installation sequence.
  Product priorities are numbers between 100 and -100 (0 is the default)

There are different possibilities how these two factors are used to determine the installation order. Therefore, opsi provides two algorithms.

Switch between these algorithms can be done either:

**using opsi-configed, in the server configuration**

![Figure 67. opsi-configed: server configuration](image)

or on the command line with the following command:

```
opsi-setup --edit-config-defaults
```
Algorithm 1: product dependency before priority (default)

With this algorithm, the products are first sorted based on their priorities and then re-sorted based on the product dependencies. This of course allows a product with a very low priority to be pushed far forward because it is required by a product other than required before. On the other hand, it prevents installation problems due to unresolved product dependencies.

Algorithm 1 ensures that the installation order is constant, regardless of how many products are set to setup. This order corresponds to the order which is shown in configured when the products are sorted according to the position column.

This ensures that if a setup script is only interrupted with "ExitWindows /immediateReboot", the processing of the interrupted script is continued immediately after the reboot.

Algorithm 2: product priority before dependency

This algorithm is based on the idea that in practice, there are essentially three priority classes:
• Products to be installed first such as OS patches and drivers that bring the PC to a standard state. Is realized by assigning a high priority (maximum +100).

• "Normal" products that install applications (default priority 0).

• Products that should be installed last, e.g. software inventory control. Realized by assigning a low priority (lowest possible -100).

Product dependencies are only resolved within a priority class. This ensures that products with a high priority are actually installed at the beginning. Cross-priority product dependencies are not taken into account or give a warning. It is therefore important to note when packaging that product dependencies are only defined within one priority class.

The product dependencies are interpreted here in such a way that with "normal" products they automatically lead to a consistent order that takes all dependencies into account. If contradictory (circular) dependencies have been defined, an error is displayed.

In the case of products with high priorities that are essential for setup of the computer, however, the administrator should set the exact order by hand - similar to, for example, Unix start-up scripts - by assigning a specific priority between +100 and +1 for each product according to the desired order. The same applies to the final products with low priorities.

**Defining product priorities and dependencies**

Priorities and product dependencies belong to the metadata of an opsi-product. You will be asked for these when creating a product with the command `opsi-newprod`.

This metadata is stored in the control file of the product and can be edited there. After a change in the control file, the product must be repacked and installed again.

See also the chapter *Creating an opsi product package* in the opsi-getting-started manual.

### 8.1.3. Integration of new software packages into the opsi software deployment.

Instructions for integrating your own software can be found in the opsi-getting-started manual.

### 8.2. Netboot products

#### 8.2.1. Parameters for the opsi linux boot image

The opsi-linux-bootimage has some parameters which may be used to change the behaviour of the bootimage. You will try this if the opsi-linux-bootimages doesn't run properly with the standard parameters on your hardware (e.g. black screen).

You may change these standard parameters by the `opsi-configed` choosing the Tab Hostparameter and use there the entry `opsi-linux-bootimage.append`.

Typical values are (may be combined):
For AMD Ryzen 2XXX Prozessoren we recommend to use the parameters:

- `mem=2G`
- `ramdisk_size=2097152`

AMD Ryzen 3XXX additionally need the parameter

- `nomodeset`

to correctly display the bootimage.

- `dhclienttimeout=SECONDS`

sets the value `timeout` in `/etc/opsi/dhclient.conf` within the opsi-linu-bootimage (default 30 seconds).

An other important default is the password of the root user within the opsi-linux-bootimage. This password is `linux123` by default and you should change this for security reasons.

To do this change the `opsi-linux-bootimage.append` entry at the `server-configuration`.

The option you have to change is `pwh` (password hash). As the value to this option you have to give a new password as a hash, which will be loaded to the `/etc/shadow` during the boot process.

The best way to get the correct password hash is to login via ssh to your bootimage:

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

The old password is `linux123`.

Now set a new password for root:

```
passwd
```

Get the new hash

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

The output should look like this:
Now copy from after the first colon until to the second colon and use this as value for pwh.

So the option for `opsi-linux-bootimage.append` may be:

```
pwh=$6$344YXKIT$D4RPZfHMmv8e1/i5nNkOfaRN2oYNobCEjCHnkehiEFA7NdkJDKW9KF4960HBmyHHq0kD2FBLHZoTdr5YoDLIoWz/:14803:0:99999:7:::
```

One can execute a python script before the execution of the desired netboot product. Therefore the bootimage append supports two parameters:

- `pre-execute`
- `pre-script`

In addition these parameter require an address with the script. This can be a `http://` or `tftp://` address. Please refer to the following example:

```
• tftp://172.16.166/linux/test.py
```

When using tftp please keep in mind that the default port 69 is used.

### 8.2.2. Unattended automated OS installation

#### 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-server.

**Using CD-Boot:** * The client boots the boot image from the `opsi-client-boot-cd`. 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 bootimage formats and partitions the hard disk. * The bootimage copies the required installation files and configuration information from the `opsi-server` to the client and initiates a reboot. * Windows Installation: After the reboot the client installs the OS according to the provided configuration information without any interaction. * Linux Installation: By default the Linux Netboot products initiate a `kexec` command from within the bootimage and directly jump to the distribution installer. * Next the `opsi-client-agent` 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.
Preconditions

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. If there is no possibility to use PXE you may boot from the opsi-client-bootcd.

The opsi installation package for the OS to be installed needs to be provided on the depot server. In the following we assume Windows 10 to be the OS to install.

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.

![Diagram of PXE boot process](image)

*Figure 70. 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 (bootimage) given in the configuration information.

**Loading pxelinux**

The bootimage 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 x/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 a multi-stage process:

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 
eternet 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.
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 `opsipxeconfd` 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.bz2`).

**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.
Please read the chapter *Create a new client using the opsi-client-bootcd* at the opsi-getting-started manual.

**The Linux bootimage prepares for reinstallation**

The bootimage again performs a dhcp request and configures the network interface according to the perceived information. Afterwards the configuration data for the client will be loaded via * OPSI Web Service*.

![Diagram](image)

*Figure 72. PXE-Boot loaded with bootimage preparing hard disk for operating system installation*

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

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

Then the password for the user *pcpatch* will be decrypted with the transferred key to mount the installation share and then call the installation script from the mounted share to start the installation of the operating system. What specific operations the script performs depends on the operating system which is to be installed. Below the steps of a Windows 10 installation will be described.

**Prepare the disc:** On the hard disk the bootimage creates a new partition (size of 4 GB), formats it and installs a bootable ntloader kernel.

**Copy the installation file:** The files required for OS installation and the setup files for the opsi-client-agent (which is the opsi software distribution pack) will be copied from the server file share (e.g. `/var/lib/opsi/depot/win10/installfiles`) 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. This is built from configuration files and the dhcp-response. For example the file *unattend.xml*, which is the control file for unattended OS Installation, will be patched with specific information like host IP, client IP, client name, workgroup, default gateway etc..

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

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 *localboot 0*. The local boot loader will be started and the setup for operating system installation starts.

These steps are controlled by an OS specific python script.

Installation of OS and opsi-client-agent

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

How the patcha program works

As mentioned above the information collected from dhcp and opsi-webservice will be used to patch some configuration files as e.g. *unattend.xml*. 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 the specified properties in the Windows 10 product. 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 -f <patchValues> <filename>` the file `<filename>` will be patched with information from the set product properties, stored in the file `<patchValues>`. The file `<patchValues>` is generated before the installation from the product properties.

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

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

**patcha** patches one tag per line

Caveat: `patch a` patches only the first pattern of each line.

Each pattern will be expanded (or reduced) to the length of the value to be replaced with and then replaced. Trailing chars will not be affected.

Examples:

With the input file `try.in`

```
cat try.in
tag1=hallohallohallo1 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:
Structure of the unattended installation products

The information about the Structure of the unattended installation products is found in the opsi-getting-started manual.

Simplified driver integration with symlinks

The information about the Simplified driver integration with symlinks is found in the opsi-getting-started manual.

8.2.3. Some hints to the NT6 netboot products (Win7 to Win 10)

Preconditions

All netboot products with the version $\geq 4.1.0.0$ require an opsi-winst $\geq 4.12.0.13$ installed on the opsi server.
The netboot products with the version 4.1.0.0 also run on opsi 4.0.7.

Multidisk mode

Windows OS installations on systems with more than one hard disk are now supported with the new multidiskmode property. The multidiskmode property allows to select the target disk for the Windows installation by selecting the disk number. It is also possible to select the first SSD by using prefer_ssd or to select the first rotational (classic) disk by prefer_rotational.

In order to work with the multidisk mode, the property winpenetworkmode has to be true.

By using the multidiskmode on a computer with MBR BIOS you have to make sure, that the by multidiskmode selected disk is also the first disk in the BIOS boot sequence.

On UEFI BIOS systems no further actions are necessary, due to the fact that the boot sequence it's controlled by the installation software.

Actions while running inside the Windows PE

The preparation of a Windows installation starts with the opsi-linux bootimage, which select and
prepares the hard disk. It also copies a Windows PE to a partition of the hard disk. This Windows PE is
booted in order to start the Windows setup.

Starting with the 4.1.0.0 netboot products we use an opsi-script inside the Windows PE. This has the
following advantages:

- Easier and clearer scripts
- The creation of a log file of the actions inside the PE
- Sending of this log file to the opsi server

**NT6 product properties**

The netboot products for the installation of the operating systems of the NT6 family, contain several
properties which will be described below.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>additional_drivers</td>
<td></td>
</tr>
<tr>
<td>administrator_password</td>
<td></td>
</tr>
<tr>
<td>asio_orientation</td>
<td>true</td>
</tr>
<tr>
<td>boot_partition_label</td>
<td>BOOT</td>
</tr>
<tr>
<td>boot_partition_letter</td>
<td></td>
</tr>
<tr>
<td>boot_partition_size</td>
<td>0</td>
</tr>
<tr>
<td>data_partition_create</td>
<td>true</td>
</tr>
<tr>
<td>data_partition_label</td>
<td>DATA</td>
</tr>
<tr>
<td>data_partition_letter</td>
<td>D</td>
</tr>
<tr>
<td>data_partition_preserve</td>
<td>never</td>
</tr>
<tr>
<td>full_name</td>
<td></td>
</tr>
<tr>
<td>imagename</td>
<td></td>
</tr>
<tr>
<td>install</td>
<td>disk</td>
</tr>
<tr>
<td>mui_disk/node</td>
<td>0</td>
</tr>
<tr>
<td>orgname</td>
<td></td>
</tr>
<tr>
<td>pre_format_system_partitions</td>
<td>true</td>
</tr>
<tr>
<td>preserve_windows_partition</td>
<td>false</td>
</tr>
<tr>
<td>product_key</td>
<td></td>
</tr>
<tr>
<td>setup_zstar_install</td>
<td></td>
</tr>
<tr>
<td>system_keyboard_layout</td>
<td>0407:000000407</td>
</tr>
<tr>
<td>system_language</td>
<td>de-DE</td>
</tr>
<tr>
<td>system_timezone</td>
<td>W. Europe Standard Time</td>
</tr>
<tr>
<td>use_raid1</td>
<td>false</td>
</tr>
<tr>
<td>windows_partition_label</td>
<td>WINDOWS</td>
</tr>
<tr>
<td>windows_partition_size</td>
<td>100%</td>
</tr>
<tr>
<td>windows_dir</td>
<td>auto</td>
</tr>
<tr>
<td>windows_winntlocale</td>
<td>0407:000000407</td>
</tr>
<tr>
<td>windows_partition_size</td>
<td>400M</td>
</tr>
<tr>
<td>windows_language</td>
<td>de-DE</td>
</tr>
<tr>
<td>windows_language_fallback</td>
<td>de-DE</td>
</tr>
<tr>
<td>winpe_netmode</td>
<td>true</td>
</tr>
</tbody>
</table>

**additional_drivers**

One or more directories below `<productid>\drivers\drivers\additional`. All driver directories
below the given directories will be integrated. If there is here a driver for a found device, no other
driver will be integrated by the automatic driver integration.

**administrator_password**

At this property you set the password for the local Administrator.

Default = nt123

**architecture**

Select the bootimage architecture (e.g. 32/64 Bit). This will not (!) affect the architecture of the
installed Operating System.

Default = 65bit Since version 4.1.0.0-15
askbeforeinst
   Should there be a confirmation dialog before start installing

boot_partition_label
   Label of the boot_partition (Bitlocker partion)

boot_partition_letter
   Drive letter of the boot_partition (Bitlocker Partion)

boot_partition_size
   Size of the boot_partition (Bitlocker Partion). 0 = create no partition

data_partition_label
   Label of the data partion (if created)

data_partition_letter
   Drive letter of the data partion (if created)

data_partition_preserve
   Preserve data partition on reinstallation

fullname
   Full name of the license holder, which is given to the setup program

imagename
   Name of the operating system variant
installto

This property should never be changed. It is not editable. It is used internally to difference between standard (disk) installations, opsi-local-image (oli) and opsi-vhd (vhd). Please do not try to change it.

multi_disk_mode

This property is used to select the target disk of the Windows installation. Possible values are: "0", "1", "2", "3", "prefer_ssd", "prefer_rotational"
The values "0", "1", "2", "3" are the index of the hard disks ("0" = 1. harddisk)
The value "prefer_ssd" selects the first SSD.
The value "prefer_rotational" selects the first rotational (classic) disk.
This property is ignored on systems with only one disk.
Default = "0"

orgname

Name of the company or organisation of license holder, which is given to the setup program

pre_format_system_partitions

Should we format the windows and boot partition before installation starts to remove any traces of former installations? (takes time!)

Figure 73. NT6 image names
**preserve_winpe_partition**

By default (False) the winpe partition will be deleted after the installation and the space is used by the system partition. True means only to hide the winpe partition.

**productkey**

License key for the installation. Is only used if the host parameter `license-management.use` is set to `false`. If it set to `True` the license key will be get from the license management module.

**setup_after_install**

Which opsi product(s) should we switch to setup after OS installation is done ?

**system_keyboard_layout**


![Select keyboard language](image)

*Figure 74. Select keyboard language*

**system_language**

Select system language

**system_timezone**

Select time zone

**winpe_dir**

This property only is used for debugging

The value "auto" detects the matching standard winpe directory. These are `winpe` or `winpe_uefi`

Any other value must point to an existing directory inside the product directory on the opsi depot share.

Default = `auto`

**winpe_inputlocale**

Microsoft-Windows-International-Core-WinPE InputLocale

**winpe_partition_size**

Size of the winpe_partition

**winpe_uilanguage**

Microsoft-Windows-International-Core-WinPE
winpe_uilanguage_fallback
Microsoft-Windows-International-Core-WinPE

windows_partition_label
Label of the system partition (c:)

windows_partition_size
Size of the system partition (c:). The size may be given as percent of the harddisk size or as absolute size (G=Gigabyte). If you choose a value other than 100%, the rest will be used as data_partition.

Figure 75. Size of the system partition

winpenetworkmode
If true the PE tries to mount the depot share and start the operating system setup from the share (faster). If false all installation files will be copied to the hard disk and the installation starts from the local disk (slower).

8.2.4. memtest

The product memtest is a utility to perform a memory test on a client.
8.2.5. hwinvent

This product does a hardware inventory of the client.

8.2.6. wipedisk

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

8.3. Inventory

The localboot products hwaudit and swaudit and the netboot product hwinvent are available for inventory purposes.

8.3.1. Hardware Inventory

The hardware inventory is controlled in opsi via a configuration file. This means that the information how and which data is collected is not hardwired into the corresponding products hwaudit and hwinvent. Rather, these products are controlled via a configuration file. For this purpose, each time an inventory product is executed, the configuration file is read in and interpreted via the opsi web service. At the same time, this configuration file also controls the structure of the database, so that an extension of this configuration file also entails an extension of the database schema.

The configuration file is /etc/opsi/hwaudit/opsihwaudit.conf. In this file all objects to be inventoried are defined and is described how the data belonging to this object is to be collected (under Linux and under Windows). At the same time, the associated data structure is also defined. For simplification, this configuration file contains inheritance mechanisms that are based on an object orientation. The background to this is the fact that many objects have identical data fields, e.g. Name and Vendor. This general information is defined in virtual hardware classes. The actual inventory objects are then structural hardware classes, which can inherit many properties from the above virtual classes.

An example to explain this mechanism:
The configuration file first defines a virtual class named "BASIC_INFO". This defines the properties (Values):

- "name"
- "description"

Next comes the virtual class named "HARDWARE_DEVICE", which inherits all the properties of "BASIC_INFO" and defines the following additional ones:

- "vendor"
- "model"
- "serialNumber"
8. opsi products

Next comes the first object, which we also find in the inventory, the first structural class name "COMPUTER_SYSTEM", which inherits all properties of "HARDWARE_DEVICE" and defines or overwrites the following additional ones:

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

Various properties are described in the context of the definition of a class and its values:

- Class definition:
  - "Type" is "STRUCTURAL" or "VIRTUAL"
  - "Super" specifies the class from which to inherit.
  - "Opsi" specifies the name of the class, which will later be used as the display name in opsi.

Furthermore, the class definition can indicate how this data is collected. This information can also be used to define the values.

- For inventory under Linux:
  - "Linux": "[<command>]<parameter>"
    Execution of the command line program <command>, with the argument <parameter>.
  - "Python": "<python code with place holder>"
    Execution of the specified Python code whereby the placeholder is then replaced by the determined values.

- For the Inventory under Windows:
  - "WMI": "<wmi select statement>"
    WMI query to be executed.
  - "Cmd": "<Python text object with place holder>"
    In this case, this is the relative path to an executable Python program, whose output replaces the placeholder.
  - "Registry": "[<registry key>] <value name>"
    The value of <value name> is read from the registry in <registry key>. The registry is read in an architecture-specific manner. This means that the 64-bit branch of the registry is read on a 64-bit system.

- Value Definition:
  - "Type": "<MySQL database type>"
    <MySQL database type> specifies the data type which will be used to create this value in the database.
"Scope": "<scope>"
The <scope> field is used in the following way:
"g" means: This attribute is the same for all devices of this type.
"i" means: This attribute can have different values for devices of this type.

"Opsi": "<id>"
$id$ is the opsi internal name of the field. This can be localized for output via the files in
/etc/opsi/hwaudit/locales.

"WMI": "<id or command>"
$id$ or $command$ is either the name of the WMI command, specified in the class definition,
which outputs the value, or a separate WMI command.

"Linux": "<id>"
$id$ is the name of the Linux command, specified in the class definition, which outputs the
value.

"Condition": "<condition>"
$condition$ is a condition which must be fulfilled, in order for the $value$ to be determined. For
example, if the $condition$ is defined as "vendor=[dD]ell*", then the values of "vendor" must
contain either Dell or dell.

Here is an example of the class "COMPUTER_SYSTEM":

The last value, "dellexpresscode", is particularly interesting:
This only makes sense if it is also a Dell computer, hence the condition.
Under Windows the command line program `dellexpresscode.exe` is executed, which is located as seen from `hwaudit.exe` in the subdirectory `dellexpresscode\`. This produces an output in the form:
**8. opsi products**

```plaintext
dellexpresscode=123456789. With the .split('=')[1] after the placeholder, the value after the equal sign is used.

Under Linux it is checked in which element (COMPUTER_SYSTEM or CHASSIS) a value was found for serialNumber and this is then used to calculate the Dell expresscode.

The opsi names of the values are translated using the files found in /etc/opsi/hwaudit/locales/*. Example of /etc/opsi/hwaudit/locales/en_US:

```plaintext
COMPUTER_SYSTEM = Computer
COMPUTER_SYSTEM.systemType = Type
```

The class name COMPUTER_SYSTEM is translated into "Computer". The opsi attribute "systemType" of the class COMPUTER_SYSTEM is translated into "type". Finally this note: If a new field is created, you should create it in the locale files, even if you do not translate the term yourself. This prevents "Warning" messages from being generated while running.

After you have modified the configuration file and the locales, you have to execute the following command so that the changes are also transferred to the database:

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

Furthermore you have to completely reload the data in opsi-configed: File / Reload all data.

The source code of this package can be found on GitHub: opsi-org/hwaudit

### 8.3.2. Software Inventory

Software inventory is done with the localboot product swaudit. The information from the uninstall path of the registry is collected and supplemented with additional information about hotfixes and license keys.

The source code of this package can be found on Github: opsi-org/swaudit

### 8.4. opsi subscriptions

#### 8.4.1. Initial Deployment of opsi subscriptions

To provide the required packages, you can download them manually or install all packages after editing the configuration of the opsi-package-updater.

#### 8.4.2. Configuration of the opsi-package-updater / updating the opsi subscriptions

For further operation it is recommended to save the repository configuration in /etc/opsi/package-
Updater: `repos.d/`. You will receive a corresponding configuration file with your access data.

⚠️ If using a proxy, the access must be configured in these files!

It then offers to only install the required packages from the new repositories:

```bash
# Example: Install packages for mshotfix, OS Windows 7 x64 / Server 2008 R2 and Windows 10 / 2016 / 2019 x64
opsi-package-updater --repo uib_abo_mshotfix install mshotfix mshotfix-win7-win2008r2-x64-glb mshotfix-win10-win2016-x64-glb
# Example: Install packages for Office 2016
opsi-package-updater --repo uib_abo_msoffice install office_2016_hotfix
# Example: Install packages for the standard opsi products: firefox and libreoffice
opsi-package-updater --repo uib_abo_standard install firefox libreoffice
```

It is possible to install only required packages, e.g. only the packages to update Windows 7:

```bash
opsi-package-updater -v install mshotfix mshotfix-win7-x86-glb mshotfix-win7-win2008r2-x64-glb
```

To update the packages the usage of `opsi-package-updater` is recommended.

By executing `opsi-package-updater -v update` you can update the installed packages, for example via a cronjob.

An equivalent is to update the packages for each repository:

```bash
# Updating packages for mshotfix
opsi-package-updater --repo uib_abo_mshotfix update
# Updating packages for msoffice
opsi-package-updater --repo uib_abo_msoffice update
# Updating packages for standard products
opsi-package-updater --repo uib_abo_standard update
```

### 8.4.3. Configuration of default properties

When installing opsi packages non-interactively, the default properties of the packages are saved as default on a `opsi-config-server` or `opsi-server`.

As of opsi version 4.0.5, the default properties of the `opsi-server` can be set via the management interface `opsi-configured`.

If you subsequently install a newer package version using `opsi-package-manager` or `opsi-package-updater`, the default properties of the depot will be retained.
8.4.4. Subscription **MS-Hotfixes**


Versions no longer supported by Microsoft are displayed as "failed": 1507 "non"-LTSB, 1511 and 1607 "non"-LTSB except Education and Enterprise versions.

The updates will be available within three working days after the release of a patch that Microsoft considers to be important or critical.


<table>
<thead>
<tr>
<th>Rating</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>A vulnerability whose exploitation could allow code execution without user interaction. These scenarios include self-propagating malware (e.g. network worms), or unavoidable common use scenarios where code execution occurs without warnings or prompts. This could mean browsing to a web page or opening email. Microsoft recommends that customers apply Critical updates immediately.</td>
</tr>
<tr>
<td>Important</td>
<td>A vulnerability whose exploitation could result in compromise of the confidentiality, integrity, or availability of user data, or of the integrity or availability of processing resources. These scenarios include common use scenarios where client is compromised with warnings or prompts regardless of the prompt's provenance, quality, or usability. Sequences of user actions that do not generate prompts or warnings are also covered. Microsoft recommends that customers apply Important updates at the earliest opportunity.</td>
</tr>
<tr>
<td>Moderate</td>
<td>Impact of the vulnerability is mitigated to a significant degree by factors such as authentication requirements or applicability only to non-default configurations. Microsoft recommends that customers consider applying the security update.</td>
</tr>
</tbody>
</table>
The opsi-mshotfix package uses (like WSUS Offline Update [http://forums.wsusoffline.net/viewtopic.php?f=7&t=172](http://forums.wsusoffline.net/viewtopic.php?f=7&t=172)) to download Microsoft’s update catalog file `wsusscn2.cab`, to identify the required patches. This catalog file contains at least all the “critical” and “security-relevant” updates, but not necessarily all the “important” and “optional” updates.

The opsi mshotfix packages have a modular structure. The basic package “mshotfix” contains only a script for installing the patches. The actual patches are contained in separate packages.

### Table 2. mshotfix Client Requirements

<table>
<thead>
<tr>
<th>OS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7 SP1 / Windows 2008 R2 SP1</td>
<td></td>
</tr>
<tr>
<td>Windows 8 / Windows 2012</td>
<td></td>
</tr>
<tr>
<td>Windows 8.1 / Windows 2012 R2</td>
<td></td>
</tr>
<tr>
<td>Windows 10 / Windows 2016 / Windows 2019</td>
<td>Warnings:</td>
</tr>
<tr>
<td>- WAN/VPN extension: The packages for Windows 8.1 / Windows 2012 R2 require opsi-winst &gt;= 4.11.3.11-1 and opsi-client-agent &gt;= 4.0.4.4-1</td>
<td></td>
</tr>
<tr>
<td>- WAN/VPN extension: The packages for Windows 10 / Windows 2016 require opsi-client-agent &gt;= 4.0.7.9-2</td>
<td></td>
</tr>
<tr>
<td>- WAN/VPN extension: The packages for Windows 10 / 2016 / 2019 require opsi-client-agent &gt;= 4.1.0.0-27 and opsiclientd &gt;= 4.0.95</td>
<td></td>
</tr>
</tbody>
</table>

Structure of the subscription area on our download server:
mshotfix
  !-opsi4/
    !-glob/ Base-package mshotfix and global packages
    mshotfix-win7-x86-glb
    mshotfix-win7-win2008r2-x64-glb
    mshotfix-win8-win2012-x64-glb
    mshotfix-win81-x86-glb
    mshotfix-win81-win2012r2-x64-glb
    mshotfix-win10-win2016-x64-glb
    mshotfix-win10-x86-glb
    mshotfix-win10-1507-x64-glb     (since 201904-1)
    mshotfix-win10-1507-x86-glb     (since 201904-1)
    mshotfix-win10-1607-x64-glb     (since 201904-1)
    mshotfix-win10-1607-x86-glb     (since 201904-1)
    mshotfix-win10-1703-x64-glb     (since 201904-1)
    mshotfix-win10-1703-x86-glb     (since 201904-1)
    mshotfix-win10-1709-x64-glb     (since 201904-1)
    mshotfix-win10-1709-x86-glb     (since 201904-1)
    mshotfix-win10-1803-x64-glb     (since 201904-1)
    mshotfix-win10-1803-x86-glb     (since 201904-1)
    mshotfix-win10-1809-x64-glb     (since 201904-1)
    mshotfix-win10-1809-x86-glb     (since 201904-1)
    mshotfix-win10-1903-x86-glb     (since 201904-1)
    mshotfix-win10-1903-x64-glb     (since 201904-1)
  !-misc/ various additional packages
    dotnetfx
    dotnetfx-hotfix
    mshotfix-uninstall
    ms-ie11
    ms-optional-fixes
    silverlight

The following table should help you choose the right packages:

**Table 3. mshotfix Client-OS**

<table>
<thead>
<tr>
<th>OS</th>
<th>Arch</th>
<th>Patch-package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7</td>
<td>32Bit</td>
<td>mshotfix-win7-x86-glb</td>
</tr>
<tr>
<td>Windows 7</td>
<td>64Bit</td>
<td>mshotfix-win7-win2008r2-x64-glb</td>
</tr>
<tr>
<td>Windows 7</td>
<td>64Bit</td>
<td>mshotfix-win8-win2012-x64-glb</td>
</tr>
<tr>
<td>Windows 8.1</td>
<td>32Bit</td>
<td>mshotfix-win81-x86-glb</td>
</tr>
<tr>
<td>Windows 8.1</td>
<td>64Bit</td>
<td>mshotfix-win81-win2012r2-x64-glb</td>
</tr>
</tbody>
</table>
### Installation:

```
opsi-package-manager -i mshotfix_201008-1.opsi
```

To set to setup wherever the product is installed:

```
opsi-package-manager -iS mshotfix_201008-1.opsi
```

In addition to the basic package, the patch packages are installed in the same way. However, since these packages do not contain any installation scripts, they can only be used together with the basic package, i.e. you cannot set them separately on setup. The mshotfix basic package is responsible for the client installation.

Since package mshotfix 201304-1, a list of patches installed by mshotfix is stored locally in the file `C:\opsi.org\mshotfix\deployed.txt`.

### Caution

Since package mshotfix 201808-3, the current Servicing Stack is installed first, followed by an immediate reboot.

**noreboot**

- `noreboot=on`: Don't Reboot if possible Warning will be logged if a reboot is required. Will be ignored for Servicing stacks values: ["off", "on"] default: ["off"]

**force**

- `force=on`: All Hotfixes will be forced installed values: ["off", "on"] default: ["off"]
excludes

Commaseparated list with kb-numbers or ms-no, that will be excluded (Only Number without beginning kb and no spaces. Example: 123456,789011,976002

severity

choose the severity that will be installed. Possible Entries are Critical, Important, Moderate, all values: ["Critical", "Important", "Moderate", "all"] default: ["Critical", "Important"]

excludelist-superseded.txt

Use File ExcludeList-superseded.txt values: ["", "ExcludeList-superseded.txt"] default: [""]

monthly-updates

Handle windows-7-and-windows-8-1: security Only Quality Update vs Monthly Quality Rollup (see Further simplifying servicing models for Windows 7 and Windows 8.1, More on Windows 7 and Windows 8.1 servicing changes, .NET Framework Monthly Rollups Explained) values: ["all", "monthly_quality_rollup", "security_only_quality_update"] default: ["security_only_quality_update"]

misc mshotfix-uninstall

| mshotfix-uninstall | 201512-1 | MS Hotfix BasePackage |

Removes patches that can be uninstalled via wusa /uninstall ....

excludes

Commaseparated list with kb-numbers or ms-no, that will be excluded (Only Number without beginning kb and no spaces. Example: 2553154,ms14-082)

noreboot

noreboot=on: Don't Reboot. Warning will be logged if a reboot is required. values: ["off", "on"] default: ["off"]

removefromdeployed.txt

Remove from deployed.txt default: False

removekb

Remove KBXXXXX, (Only Number without beginning kb and no spaces f.e. 3097877) multivalue: True default: [""]

misc dotnetfx

| dotnetfx | 4.7.2-3 | .NET Framework |

Package for installing NET Framework versions 4.5 and higher, especially on Windows 7 / 2008 R2 /
8.1 / 2012 R2 / 10 NET 3.5 can also be installed on Windows installations (newer than Windows 7).

**version**

The Microsoft .NET Framework 4.8 is a highly compatible, in-place update to the Microsoft .NET Framework 4, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1 and 4.7.2

values: ["3.5", "4.0", "4.5", "4.5.1", "4.5.2", "4.6", "4.6.1", "4.6.2", "4.7", "4.7.1", "4.7.2", "latest", "latestAnd3.5"]

default: ["latest"]

rerundotnethotfix::rerun dotnetfx-hotfix after installation if possible values: ["false", "true"] default: ["true"]

**install_language_languagepack**

install_language_languagepack values: ["auto", "de", "en", "fr"] default: ["auto"]

**os-package**

Here you can switch from which OS-Version to be install Dotnet3.5, auto=win10 or opsi-local-image-win10 (default); other ProductID for netboot-product values: ["auto", ] default: ["auto"]

**misc dotnetfx-hotfix**

<table>
<thead>
<tr>
<th>dotnetfx-hotfix</th>
<th>201808-1</th>
<th>dotnetfx-hotfix</th>
</tr>
</thead>
</table>

The MS-Hotfixes update subscription only contains the hotfixes for Microsoft .NET Framework that are applicable for the respective operating system. E.g. Windows 7 came with "Microsoft .NET Framework 3.5.1"

(However, since October 2016 there have also been irregular "Monthly Rollups" for DotnetFramework, which are included in the mshotfix package.)

The dotnetfx-hotfix package contains the hotfixes from Microsoft for

- Microsoft .NET Framework 4 and higher for Windows 7

The package dotnetfx-hotfix is currently patching versions 4.x - if available - to the newest version of the series.


respectivly

https://support.microsoft.com/en-us/lifecycle#gp/Framework_Faq

**noreboot**

noreboot=on: Don’t Reboot. Warning will be logged if a reboot is required. values: ["off", "on"]
default: ["off"]
8. opsi products

force

force=on: All Hotfixes will be installed forced values: ["off", "on"] default: ["off"]

severity

choose the severity that will be installed. Possible Entries: Critical, Important, Moderate, all values: ["Critical", "Important", "Moderate", "all"] default: ["all"]

misc ms-ie11

<table>
<thead>
<tr>
<th>ms-ie11</th>
<th>11.0-11</th>
<th>Internet Explorer 11</th>
</tr>
</thead>
</table>

Win7 Internet Explorer 11

rerunmshotfix

rerun mshotfix after installation values: ["false", "true"] default: ["true"]

client_language

values: ["auto", "de", "en", "fr"] default: ["auto"]

misc ms-optional-fixes

<table>
<thead>
<tr>
<th>ms-optional-fixes</th>
<th>201808-1</th>
<th>MS optional fixes</th>
</tr>
</thead>
</table>

Intended as a supplement to mshotfix and based on mshotfix.

Contains:

kb2999226 win7-win8.1

message_language

Language for messages while installing values: ["auto", "de", "en", "fr"] default: ["auto"]

noreboot

noreboot=on: Don't Reboot if possible Warning will be logged if a reboot is required. Will be ignored for Servicing stacks values: ["off", "on"] default: ["off"]

excludes

Comma-separated list with kb-numbers or ms-no, that will be excluded (Only Number without beginning kb and no spaces. Example: 123456,789011,2222)

rerunmshotfix

rerun mshotfix after installation values: ["false", "true"] default: ["true"]

client_language

values: ["auto", "de", "en", "fr"] default: ["auto"]
8.4.5. Update subscription for *MS-Office Hotfixes*

Regular updates for MS Office 2010/2013/2016 32 bit and MS Office 2016 64 bit.

The updates will be available within three working days after the release of a patch that Microsoft considers to be important or critical.

*Table 4. Office hotfix Requirements*

<table>
<thead>
<tr>
<th>Office Version</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office 2010 32-bit</td>
<td>Service Pack 2</td>
</tr>
<tr>
<td>Office 2013 32-bit</td>
<td>Servicepack 1</td>
</tr>
<tr>
<td>Office 2016</td>
<td></td>
</tr>
</tbody>
</table>

**Updates for MS Office 2010 32-bit international: office_2010_hotfix**

<table>
<thead>
<tr>
<th>office_2010_hotfix</th>
<th>201808-1</th>
<th>Microsoft Office 2010</th>
</tr>
</thead>
</table>


Since package office_2010_hotfix 201305-2, a list of patches installed by office_2010_hotfix is stored locally in the file *C:\opsi.org\mshotfix\office_2010_hotfix_deployed.txt*.

Since office_2010_hotfix 201503-1:

**excludes**

Comma separated list with kb-numbers or ms-no, that will be excluded (Only Number without beginning kb and no spaces. Example: 2553154,ms14-082)

**Updates for MS Office 2013 32-bit international: office_2013_hotfix**

<table>
<thead>
<tr>
<th>office_2013_hotfix</th>
<th>201808-1</th>
<th>Microsoft Office 2013</th>
</tr>
</thead>
</table>


Is tested with Office 2013 Professional
A list of patches installed by office_2013_hotfix is stored locally in the file C:\opsi.org\mshotfix\office_2013_hotfix_deployed.txt.

Since office_2013_hotfix 201503-1:

excludes

Comma separated list with kb-numbers or ms-no, that will be excluded (Only Number without beginning kb and no spaces. Example: 2553154,ms14-082)

Updates for MS Office 2016 32-bit international: office_2016_hotfix

<table>
<thead>
<tr>
<th>office_2016_hotfix</th>
<th>201808-1</th>
<th>Microsoft Office 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotfixes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Contains language-independent monthly Office 2016 hotfixes (including Visio 2016).

Is tested with Office 2016 Professional

A list of patches installed by office_2016_hotfix is stored locally in the file C:\opsi.org\mshotfix\office_2016_hotfix_deployed.txt.

excludes

Comma separated list with kb-numbers or ms-no, that will be excluded (Only Number without beginning kb and no spaces. Example: 2553154,ms14-082)

CAUTION

If you want to install updates for MS Office 2016 32-bit and 64-bit, or only 64-bit:

Configure the repo uib_abo_msoffice update section: x3264 / x64

8.4.6. Update subscription for the opsi standard packages

Regular product updates for:

Adobe Reader DC Classic / Continuous (international, 32 Bit)
Adobe Flashplayer (international, 32 Bit / 64 Bit)
Apache OpenOffice.org (German, 32 Bit)
Google Chromium for business (international, 32 Bit / 64 Bit)
LibreOffice (international, 32 Bit / 64 Bit)
Mozilla Firefox (German, English, French and Dutch, 32 Bit), respectively as 32/64 Bit package since 201706
Mozilla Thunderbird (German, English and French, 32 Bit)
Java VM (javavm) based on Open JDK LTS 11 (international, 64 Bit)
Java VM 8 (javavm8) based on Open JDK LTS 8 (international, 32 Bit / 64 Bit)
Java VM (javavm-oracle-jdk) OpenJDK based on the actual open jdk implementation (international, 64 Bit)
Depending on contract, we also provide the following languages as subscriptions:

Mozilla Firefox (additionally in Czech, Danish, Italian, Norwegian or Spanish, 32 Bit)
Mozilla Thunderbird (additionally in Italian, 32 Bit)

Other languages on request.

The update will be available within two working weeks after the release of an update of these products. For security updates classified as critical by the manufacturer within one working week.

**Customizing with central configuration files**

For the opsi-packages

- adobe.reader.dc.classic
- adobe.reader.dc.continuous
- firefox
- flashplayer
- thunderbird

there is the possibility to create your own configurations and store them in the directory `custom`, which can be selected via properties. (See below for details)

**Customizing with preinst/postinst-scripts**

For the opsi-packages

- adobe.reader.dc.classic
- firefox (since 17.0.6esrorstandard-1)
- flashplayer (since 13.0.0.182or11.7.700.275-1)
- google-chrome-for-business
- javavm (since 1.7.0.51-4 )
- javavm8
- javavm-oracle-jdk
- libreoffice (since 4.3.5or4.4.0-2)
- ooffice (since 4.1.1-2)
- thunderbird (since 17.0.6esrorstandard-1)

there is the possibility to store your own custom scripts in the directory `custom\scripts`.

Simple templates for the supported scripts can be found in the directory `opsi\scripts`.
custom.actions.post.setup
custom.actions.post.uninstall
custom.actions.pre.setup
custom.actions.pre.uninstall
custom.declarations
custom.sections

custom scripts will be included in
- setup-script
- uninstall-script

custom pre-scripts will be included in
- setup-script
- uninstall-script

custom post-scripts will be included in
- in setup-script
- uninstall-script

custom.declarations
; intended for declaration of custom Variables and Stringlist Variables
; will be included with "include_insert" at top of [actions]
; but after GetProductProperties

custom.sections
; intended for declaration of custom secondary sections
; will be included with "include_append" at top of [actions]
; but after GetProductProperties

custom.actions.pre.setup (or custom.actions.pre.uninstall)
; will be included with "include_insert" at top of [actions]
; (but after GetProductProperties)

custom.actions.post.setup (or custom.actions.post.uninstall)
; will be included with "include_insert" in case of successful installation before
"endof" "actions"
; in setup-script ( or uninstall-script)


Adobe Acrobat
Reader DC classic

The adobe.reader.dc.classic-Package contains Adobe Acrobat Document Cloud Classic (MUI-Version)

Adaptation in the transform file *.mst
The Adobe Reader package from uib uses a standard transform file created with the Adobe Customization Wizard. Additionally, custom MST files can be stored in the directory `/var/lib/opsi/depot/adobe.reader.dc.classic/custom` on the share `opsi_depot`. These can be selected via this property (after reinstalling package using `opi-package-manager -i <adobe-package>`).

When importing the `adobe.reader.dc.classic` package on the opsi server, the custom directory is retained using a preinst/postinst-script.

**opsi WAN/VPN extension:** The package must be reinstalled with `opi-package-manager` after changes in the `custom` directory, so that the file `<productid>.files` is updated.

**client_language**


**classicversion**


**disableadobeacrobatupdateservice**

Disable Adobe Acrobat Update Service and remove Adobe Acrobat Update Task values: ["false", "true"] default: ["false"]
removebeforesetup
remove target version of adobe reader before install values: ["false", "true"] default: ["false"]

noreboot
description: noreboot=true: Don't Reboot. Warning will be logged if a reboot is required. values: ["false", "true"] default: ["false"]

Adobe Acrobat Document Cloud Continuous : adobe.reader.dc.continuous

The adobe.reader.dc.classic-Package contains Adobe Acrobat Document Cloud Continuous (MUI-Version)

Adaptation in the transform file *.mst

cat transform.txt
Changes vs default the transform file *.mst

Personalization Options
Suppress Eula

Installation Options
activated - Make Reader the default PDF viewer
IF REBOOT REQUIRED - suppress reboot

Shortcuts
deactivated - Destination Computer/Desktop/Adobe Reader (Icon)

Online and Acrobat.com Features
Online Features
activated - Disable product updates
Enable & Ask before Installing - Load trusted root certificates from Adobe

Online Services and Features
disable product updates
Load trusted root certificates from Adobe disable
DISABLE all Services

adobereader.mst
The Adobe Reader package from uib uses a standard transform file created with the Adobe Customization Wizard. Additionally, custom MST files can be stored in the directory
When importing the adobe.reader.dc.continuous package on the opsi server, the custom directory is retained using a preinst/postinst-script.

opsi WAN/VPN extension: The package must be reinstalled with opsi-package-manager after changes in the custom directory, so that the file <productid>.files is updated.

client_language

disableadobeacrobatupdateservice
Disable Adobe Acrobat Update Service and remove Adobe Acrobat Update Task values: ["false", "true"] default: ["false"]

removebeforesetup
remove target version of adobe reader before install values: ["false", "true"] default: ["false"]

noreboot
description: noreboot=true: Don't Reboot. Warning will be logged if a reboot is required. values: ["false", "true"] default: ["false"]

Adobe Flashplayer : flashplayer

| flashplayer | 32.0.0.171-1 | Adobe Flashplayer |

The flashplayer package contains Adobe Flashplayer in the version supported by Adobe.

flashplayer-version
values: esr, standard default: standard

The central configuration file mms.cfg is generated with every new installation and adjusted according to the product properties (exception: use of a custom mms.cfg) No entries are made for empty values. Since version 13.0.0.250-2 there has been the option to provide a custom mms.cfg

mms.cfg
A custom mms.cfg can be stored in the directory /var/lib/opsi/depot/flashplayer/custom on the share opsi_depot. This can be selected via this property (after reinstalling package using opsi-package-manager -i <adobe-package>).

When importing the flashplayer package on the opsi server, the custom directory is retained using a
opsi WAN/VPN extension: The package must be reinstalled with opsi-package-manager after changes in the custom directory, so that the file <productid>.files is updated.

When using a custom mms.cfg other settings apart from using the autoupdatedisable property are not used.

For further information see:

*Adobe Flashplayer Admin Guide:*

```plaintext
flash_player_11_1_admin_guide.pdf (flash_player_admin_guide.pdf)

# Chapter 4: Administration

You can create and place files on the end user's machine to manage features related to security, privacy, use of disk space, etc.

Privacy and security settings (mms.cfg)

As a network administrator, you can install Flash Player across the environment while enforcing some common global security and privacy settings (supported with installation-time configuration choices). To do this, you need install a file named mms.cfg on each client machine.

The mms.cfg file is a text file. When Flash Player starts, it reads its settings from this file, and uses them to manage functionality as described in the following sections.

mms.cfg file location

Assuming a default Windows installation, Flash Player looks for the mms.cfg file in the following system directories:

32-bit Windows - %WINDIR%\System32\Macromed\Flash

64-bit Windows - %WINDIR%\SysWow64\Macromed\Flash

Note: The %WINDIR% location represents the Windows system directory, such as C:\\WINDOWS.
```
assetcachesize

description: hard limit, in MB, on the amount of local storage that Flash Player uses for the storage of common Flash components
values: ["20"]
default: ["20"]

autoupdatedisable

description: Lets you prevent Flash Player from automatically checking for and installing updated versions.
values: ["0", "1"]
default: ["1"]

autoupdateinterval

description: (without meaning if AutoUpdateDisable=1) how often to check for an updated version of Flash Player

avhardwaredisable

description: Lets you prevent SWF files from accessing webcams or microphones
values: ["0", "1"]
default: ["1"]

disabledevicefontenumeration

description: Lets you prevent information on installed fonts from being displayed.
values: ["0", "1"]
default: ["1"]

fullscreendisable

description: Lets you disable SWF files playing via a browser plug-in from being displayed in full-screen mode.
values: ["0", "1"]
default: ["1"]

localfilereaddisable

description: Lets you prevent local SWF files from having read access to files on local hard drives
values: ["0", "1"]
default: ["0"]

filedownloaddisabes

description: Lets you prevent the ActionScript FileReference API from performing file downloads
values: ["0", "1"]
default: ["0"]

fileuploaddisable

description: Lets you prevent the ActionScript FileReference API from performing file uploads
values: ["0", "1"]
default: ["0"]

disableproductdownload

description: Lets you prevent native code applications that are digitally signed and delivered by Adobe from being downloaded
values: ["0", "1"]
default: ["1"]

disablesockets

description: enable or disable the use of the Socket.connect() and XMLSocket.connect() methods
values: ["0", "1"]
default: ["1"]

enablesocketssto

description: Lets you create a whitelist of servers to which socket connections are allowed
enforcelocalsecurityinactivexhostapp

description: Lets you enforce local security rules for a specified application.

legacydomainmatching

description: Lets you specify whether SWF files produced for Flash Player 6 and earlier can execute an operation that has been restricted in a newer version of Flash Player values: ["0", "1"] default: ["0"]

localfilelegacyaction

description: Lets you specify how Flash Player determines whether to execute certain local SWF files that were originally produced for Flash Player 7 and earlier values: ["0", "1"] default: ["0"]

allowuserlocaltrust

description: Lets you prevent users from designating any files on local file systems values: ["0", "1"] default: ["0"]

localstoragelimit

description: Lets you specify a hard limit on the amount of local storage that Flash Player uses (per domain) for persistent shared objects. values: ["3"] default: ["3"]

overridegpuvalidation

description: Overrides validation of the requirements needed to implement GPU compositing values: ["0", "1"] default: ["0"]

rtmfpp2pdisable

description: Specifies how the NetStream constructor connects to a server when a value is specified for peerID, the second parameter passed to the constructor values: ["0", "1"] default: ["1"]

disablenetworkandfilesysteminhostapp

description: 1

(Acrobat.exe, Acroread.exe, WINWORD.EXE, EXCEL.EXE, POWERPNT.EXE, PPTVIEW.EXE, OUTLOOK.EXE, MSACCESS.EXE, VISIO.EXE, thunderbird.exe) values: ["0", "1"] default: ["1"]

• Known problems:
  ◦ Installation via "On Demand" may fail due to running browsers.

Google Chromium for Business

google-chrome-for-business 73.0.3683.103-1

This package contains the msi installer from Google (see Chrome for Business FAQ https://support.google.com/chrome/a/answer/188447?hl=en )

remarks:
The uninstallation and installation of google-chrome.msi sometimes fails.

Therefore there are different approaches in the opsi package to increase the reliability of the installation.

One customer reported a success rate of 100% for 40 installations with the following property settings:

- `install_architecture`: 32
- `reboot_on_retry`: True
- `reboot_after_uninstall`: True
- `timeout`: 240

In our internal tests we use: * `install_architecture`: system specific * `reboot_on_retry`: True * `reboot_after_uninstall`: True * `timeout`: notimeout

**autoupdate**

!!! Will not work anymore!!!

[https://support.google.com/chrome/a/answer/187207](https://support.google.com/chrome/a/answer/187207)

ADM= use Policy based on Googles Template, 0=UpdatesDisabled, 1=UpdatesEnabled, 2=ManualUpdatesOnly, 3=AutomaticUpdatesOnly, values: ["0", "1", "2", "3", "ADM"] default: ["0"]

**disablescheduledtaskgoogleupdatemachine**

Disable scheduled task GoogleUpdateTaskMachineCore and GoogleUpdateTaskMachineUA values: ["false", "true"] default: ["true"]

**removeupdatehelper**

default: ["true"]

**install_architecture**

description: which architecture (32/64 bit) has to be installed values: ["32", "64", "system specific"]
default: ["system specific"]

**reboot_on_retry**

description: If installation fails and (timeout > 0) then reboot before retry default: False

**reboot_after_uninstall**

description: reboot after uninstall old version default: False

**timeout**

description: TimeOutSeconds msi installs values: ["240", "300", "600", "notimeout"] default: ["notimeout"]
Apache OpenOffice : ooffice4

| ooffice        | 4.1.6-1 | Apache OpenOffice |

The ooffice package contains Apache OpenOffice in German.

handle_excel_xls
Should OpenOffice be used as the default for MS Excel documents? on = OpenOffice.org will be used as the default for MS Excel documents off = OpenOffice.org will not be used as the default for MS Excel documents

handle_powerpoint_ppt
Should OpenOffice be used as the default for MS Powerpoint documents? on = OpenOffice.org will be used as the default for MS Powerpoint documents off = OpenOffice.org will not be used as the default for MS Powerpoint documents

handle_word_doc
Should OpenOffice be used as the default for MS Word documents? on = OpenOffice.org will be used as the default for MS Word documents off = OpenOffice.org will not be used as the default for MS Word documents

LibreOffice The Document Foundation : libreoffice

| libreoffice    | 6.0.6or6.1.0-2 | LibreOffice |

The libreoffice package contains LibreOffice international.

client_language
client_language - only for messages important, because libre office is international values: ["auto", "de", "en", "fr"]

msoregister
Open Microsoft Office documents with LibreOffice (true) values: ["false", "true"] default: ["false"]

libreoffice-version
description: Stable - is an Extended Support Release from LibreOffice for the conservative user - default version (6.0.6); Experimental is a version for the experimental user from LibreOffice (6.1.0) values: ["experimental", "stable"] default: ["stable"]

hide_component
description: Hide component base by removing desktoplink and exe file values: ["base", "none"] default: ["none"]
ui_languages

description: which UI languages should be installed (comma separated), For example
UI_LANGS=en_US,de,fr,hu will install English (US), German, French, and Hungarian. default:
["auto"]

install_architecture

which architecture (32/64 bit) has to be installed values: ["32", "64", "system specific"] default:
["32"]

Mozilla Firefox : firefox

| firefox       | 60.6.1esror66.0.3-1 |

The firefox-package contains Mozilla Firefox in German, English, French and Dutch.

All by Mozilla supported versions are provided.

Firefox can be configured centrally a) either via a central configuration file mozilla.cfg (see
http://kb.mozillazine.org/Locking_preferences) b) or via a policies.json (see https://github.com/
mozilla/policy-templates/blob/master/README.md)

which must be placed in the following directory:

/var/lib/opsi/depot/firefox/custom/

When the package is installed again using opsi-package-manager -i <firefox-paket>, the configuration
files found are retained via the preinst/postinst-mechanism and can then be selected using the
"mozillacfg" property.

• example:

```
cat /var/lib/opsi/depot/firefox/custom/mozilla.cfg
//
lockPref("browser.startup.homepage", "http://www.uib.de");
lockPref("network.proxy.type", 1);
lockPref("network.proxy.http", "router.uib.local");
lockPref("network.proxy.http_port", 3128);
```

As an alternative to a mozilla.cfg, you can also integrate an autoconfig.zip created with the CCK2
using the property "mozillacfg".
opsi WAN/VPN extension: The package must be reinstalled with opsi-package-manager after changes in the custom directory, so that the file <productid>.files is updated.

**client_language**
values: ["auto", "de", "en", "fr", "nl"] default: ["auto"]

**firefox-version**
Firefox esr - Extended Support Release from Mozilla.org, standard is the newest version values: ["esr", "standard"] default: ["esr"]

**commandlineoptions**
starts firefox with the supplied parameters. See https://developer.mozilla.org/en-US/docs/Mozilla/Command_Line_Options values: ["", "-no-remote -Profile H:\ProfileDir"] default: ["""]

**pref_file**
in which file should the settings be made, user.js or prefs.js. values: ["prefs", "user"] default: ["prefs"]

**noautoupdate**
(on/off): disable auto update. default=on

**setproxy**
(off/direct/manual/file) proxy settings
- off= do nothing
- direct = direct connection to the internet
- manual = use proxy settings via property proxysetting (<ip-address>:<port>) and property noproxy_hosts (host1,host2)
- file = use proxy settings via property proxysetting (<path_to_proxyconf.pac>) and property noproxy_hosts (host1,host2)
- system
- default=off

**proxysetting**
string for proxy setting (see: setproxy)

**noproxy_hosts**
comma separated list of hosts

**mozillacfg**
derscription: filename for mozilla.cfg in %scriptpath%\custom-directory, http://kb.mozillazine.org/Locking_preferences
profilemigrator

   enable or disable Profilemigrator on first run values: ["off", "on"] default: ["off"]

   • Known problems:
     ◦ Installation via "On Demand" may fail if Firefox is running.

Mozilla Thunderbird: thunderbird

| thunderbird | 60.6.1-1 |

The thunderbird-package contains Mozilla Thunderbird in German, English and French.

All by Mozilla supported versions are provided.

Similar to the Firefox package, a central configuration file can be provided.

client_language

   values: ["auto", "de", "en", "fr"] default: ["auto"]

thunderbird-version

   values: ["60.x"] default: ["60.x"]

commandlineoptions

   starts thunderbird with the supplied parameters. See https://developer.mozilla.org/en-US/docs/Mozilla/Command_Line_Options values: ["", "-no-remote -Profile H:\ProfileDir"] default: ["]

addonsactivation

   description: Enable/Disable AddOns (default = enable) values: ["off", "on"] default: ["on"]

https://developer.mozilla.org/en/Addons/Add-on_Manager/AddonManager

http://mike.kaply.com/2012/02/09/integrating-add-ons-into-firefox/

Thunderbird_Preferences_Relevant_to_Enterprises

   Set_Netscape_User_Pref ("extensions.autoDisableScopes", 11)
   Set_Netscape_User_Pref ("extensions.shownSelectionUI", true)

enigmail

   description: Install GnuPG-Plugin values: ["off", "on"] default: ["off"]

noautoupdate

   description: disable automatic updates values: ["off", "on"] default: ["on"]
mozillacfg

description: filename for mozilla.cfg in %scriptpath%\custom-directory, http://kb.mozillazine.org/locking_preferences

lightning

description: Install calendar plugin lightning values: ["off", "on"] default: ["off"]

- **Known problems:**
  - Installation via "On Demand" may fail if thunderbird is running.

OpenJDK 11 LTS : javavm

<table>
<thead>
<tr>
<th>javavm</th>
<th>11.0.3-1</th>
<th>JDK 11</th>
</tr>
</thead>
</table>

The javavm-package contains the Open JDK 11 LTS implementations from SAP (SapMachine) and Amazon (Amazon Corretto) because Oracle no longer provides a freely available Java runtime (since early January 2019).

(Oracle announced "End Of Public Updates February 2019" http://www.oracle.com/technetwork/java/eol-135779.html)

install_architecture

description: which architecture (32/64 bit) has to be installed values: ["64 only"] default: ["64 only"]

javaversion

which version has to be installed jdk Sapmachine or amazon corretto values: ["jdkamazoncorretto", "jdksap"] default: ["jdksap"]

keepversion

description: Don't uninstall jre version values: ["1.6.0_0", "1.6.0_1", "1.6.0_2", "1.6.0_3", "1.6.0_4", "1.6.0_5", "1.6.0_6", "1.6.0_7", "none"] default: ["none"]

uninstalljava16

description: Uninstall Java 1.6 "Patch in Place" Installations default: True

uninstalljava17

description: Uninstall Java 1.7 "Patch in Place" Installations default: True

environment_set_java_home

Set the environment variable JAVA_HOME default: False

OpenJDK 8 LTS : javavm8

<table>
<thead>
<tr>
<th>javavm8</th>
<th>1.8.0.212or1.8.0.201-1</th>
<th>JDK 8 LTS</th>
</tr>
</thead>
</table>

OpenJDK 8 LTS : javavm8

<table>
<thead>
<tr>
<th>javavm8</th>
<th>1.8.0.212or1.8.0.201-1</th>
<th>JDK 8 LTS</th>
</tr>
</thead>
</table>
The javam8 package contains the Open JDK 8 LTS implementations from Amazon (Amazon Corretto) and ojdkbuild (https://github.com/ojdkbuild/ojdkbuild). The latter contains a webstart-component based on Iced Tea.

`javaversion`
which version has to be installed amazon 212 or ojdkbuild 201 values: ["jdkamazoncorretto", "ojdkbuild"] default: ["jdkamazoncorretto"]

`install_architecture`
which architecture (32/64 bit) has to be installed values: ["32 only", "64 only", "both", "system specific"] default: ["system specific"]

`keepversion`
description: Don't uninstall jre version values: ["1.6.0_0", "1.6.0_1", "1.6.0_2", "1.6.0_3", "1.6.0_4", "1.6.0_5", "1.6.0_6", "1.6.0_7", "none"] default: ["none"]

`uninstalljava16`
description: Uninstall Java 1.6 "Patch in Place" Installations default: True

`uninstalljava17`
description: Uninstall Java 1.7 "Patch in Place" Installations default: True

`environment_set_java_home`
Set the environment variable JAVA_HOME default: False

**OpenJDK : javavm-oracle-jdk**

```
| javavm-oracle-jdk | 12.0.1-1 | Open JDK |
```

The javavm-oracle-jdk-package contains the current Open JDK implementation from Oracle http://jdk.java.net/

`install_architecture`
description: which architecture (32/64 bit) has to be installed values: ["64 only"] default: ["64 only"]

`keepversion`
description: Don't uninstall jre version values: ["1.6.0_0", "1.6.0_1", "1.6.0_2", "1.6.0_3", "1.6.0_4", "1.6.0_5", "1.6.0_6", "1.6.0_7", "none"] default: ["none"]

`uninstalljava16`
description: Uninstall Java 1.6 "Patch in Place" Installations default: True

`uninstalljava17`
description: Uninstall Java 1.7 "Patch in Place" Installations default: True
environment_set_java_home

Set the environment variable JAVA_HOME default: False
9. opsi Extensions

9.1. Activation of non-free modules

Even if opsi is open source, there are some additional components that were created or maintained as part of a co-funding project and are not (yet) free of charge.

At the moment (December 2016) these are:

- MySQL backend for configuration data (see mysql backend for configuration data)
- UEFI support (see opsi with UEFI)
- opsi license management module (see opsi License Management)
- Support for clients over WAN/VPN (see opsi WAN/VPN extension)
- opsi WIM Capture (see opsi WIM Capture)
- opsi local image (see opsi local image) and opsi-vhd-reset (see opsi vhd reset)
- opsi Linux agent (see opsi Linux Support)
- opsi Nagios connector (see opsi Nagios Connector)
- User roles (see Management of user rights and roles)
- the Scalability1 extension to increase performance in very large installations

You can find more information on this topic at opsi cofunding projects.

As long as the additional extensions are in the cofunding state, they can only be used freely for evaluation purposes, but for permanent and normal use the cofunding contributions must be paid.

Which modules are allowed is specified on the opsi server in the activation file /etc/opsi/modules. It is a simple text file with information about which module is activated (for what number of clients). This file is protected against changes with a digital signature. If no information is given, the default values are used. If the activation file does not exist, only the default free components from opsi are available. In the case of temporary activation, the expiration date is included in the file.

In order to receive a temporary activation file for evaluation purposes, please contact info@uib.de. As part of participation in the corresponding cofunding projects, you will receive an activation file for the permanent and normal use of the activated components.

When you have received a modules-file, copy it to /etc/opsi.

Then run the following command to set the correct access rights:

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

Then restart opsiconfd.
Check the activation using one of the following methods:

In opsi-configed you can display the status of your activation via the menu item *Help / opsi modules*.

![Image of activation state in opsi-configed](image)

**Figure 76. Display of activation state in opsi-configed**

Using the method `backend_info` you can check with opsi-admin which modules are activated.

```
$ opsi-admin -d method backend_info
{
   "opsiVersion" : "4.0.1",
   "modules": {
      "customer": "uib GmbH",
      "vista": true,
      "vpn": true,
      "treeview": true,
      "customer": "uib GmbH",
      "license_management": true,
      "expires": "never",
      "valid": true,
      "multiplex": true,
      "signature": "THIS-IS-NO-VALID-SIGNATURE",
      "treeview": true,
      "mysql_backend": true
   }
}
```

9.2. User roles (via opsi-configed)

The feature *user roles* must be activated in the *modules* file. The functioning is explained in Section 4.16.1, “Management of user rights and roles”
9.3. opsi directory connector

9.3.1. Introduction

The opsi directory connector is a tool to transfer data from a directory service to an opsi installation. This avoids the need of maintaining data in different systems.

9.3.2. Prerequisites

This module is currently a co-funded opsi extension. There are a number of prerequisites for using this module. This means that you need an activation file. You will receive this activation when you buy the extension. For evaluation purposes, we also provide you with a temporary activation free of charge (mail to info@uib.de).

General Requirements

The source directory service must implement the LDAP protocol.

The target opsi server should run at least opsi 4.0.7. Older versions may work, but have not been tested.

The machine running the connector must have access to the directory and opsi server via the network. It is possible to run all components on the same machine, but it is assumed that separate machines are used for each component.

Hardware Requirements

These requirements are intended for basic use in a small environment with up to 500 clients. These requirements may be greater in large environments, which is why adjustments may be necessary.

- 256 MB of free memory
- Network connections

Software Requirements

The installation and operation is only supported on Linux. Support for Windows is not planned.

By using standardized protocols for communication, no additional opsi- or directory service-specific components are required.

9.3.3. Installation

To install the connector please add the opsi repository as described in the Getting Started document.

Then use the package manager of the operating system to install the package opsi-directory-connector.
On a Debian-based machine the installation can be done as follows:

```bash
apt-get install opsi-directory-connector
```

### 9.3.4. Configuration

The connector uses a number of configuration options and can be adapted to a wide variety of environments.

The configuration is done via a configuration file in JSON format, which must contain valid JSON. To specify Boolean values, please use `true` or `false`. Text has to be entered with double quotation marks, for example "this is text".

An example configuration is provided as `/user/share/opsi-directory-connector/opsi-directory-connector.example.conf.conf` This file can be used as a template for your own configuration.

```bash
cp `/user/share/opsi-directory-connector/opsi-directory-connector.example.conf.conf
/etc/opsi/opsi-directory-connector-custom.conf`
```

**Directory settings**

These settings are required to establish a connection to the directory service and to limit the search scope to specific areas and objects.

```json
{
    "directory": {
        "address": "ldap://192.168.12.34",
        "user": "DOMAIN\opsiconnector",
        "password": "insertpasswordhere",
        "passwordFile": "",
        "search_base": "dc=testcompany,dc=local",
        "search_query_computers": "(objectClass=computer)",
        "identifying_attribute": "dn",
        "connection_options": {
            "paged_search_limit": 768
        }
    },
    ...
}
```

Under **address** you have to specify at which address the server can be contacted. If the protocol **ldaps** or the port **636** is used a SSL connection to the directory is opened. Example:
User and password are used for authentication to the directory. If a value is given for passwordFile, this is interpreted as the path to a file that contains the password. The content of this file will be used as the password. This means that the password does not have to be stored in plain text in the configuration file. This will override the value set for password, if the file can be read.

We recommend using a dedicated user account.

Depending on the directory software used and its configuration, the format for the username can be different.

Besides Down-Level Logon Name in the style of DOMAIN\username, the format can also be User Principal Name in the style of user@domain, or a Distinguished Name (DN) like uid=opsiconnect,cn=users,dc=test,dc=intranet.

With search_base you specify from which point the connector looks for matching entries. The filter used to search for clients can be configured via search_query_computers.

Through the optional parameter identifying_attribute it is possible to set which attribute should be used for the unique identification of a client. This is possible since version 23. The standard used here is dn. A common alternative to dn is the value distinguishedName, this variant is often used in Microsoft Active Directory.

The connection_options parameter contains additional options for configuring the connection. The verify parameter determines whether or not the certificate should be verified for an SSL connection. In addition, the path to a CA file can be specified. If a file is specified, verification is performed via this file. With start_tls it can be controlled whether a secured connection is started after logging in to the directory.

If the optional parameter paged_search_limit is present and if its value is an integer, multiple queries are used to read the elements from the directory. The maximum number of elements an answer contains is controlled by this value. This behavior has been supported since version 20.

Additional connection options will be implemented on request.

Since version 14 it is possible to test the connection to the directory via the parameter --check-directory, without establishing a connection to the opsi server.

Connection to Univention Corporate Server

A complete Distinguished Name must be used as the user name for a connection to Univention Corporate Server (UCS). This has the form uid=<username>,cn=users,dc=company,dc=mydomain.

On UCS, LDAP can be reached via ports 7389 (unsecured) or 7636 (SSL-secured). If Samba is also installed on the server and set up as an AD-compatible domain controller, it listens on ports 389
(unsecured) or 636 (SSL-secured). To use the SSL-secured ports, the connection option `start_tls` must be set to `true`.

These two possible connections also differ in the type of login name. With LDAP, `uid=…` is used, whereas with Samba, `dn=…` is used.

Generally, you will search for computer objects in the `computers` container. The following command outputs the appropriate value for `search_base`:

```bash
echo "cn=computers,$(ucr get ldap/base)"
```

To search for Windows clients you can set `search_query_computers` to `(objectClass=univentionWindows)`.

How to create a user with read-only access can be found in the Univention Wiki: [Cool Solution - LDAP search user](http://wiki.univention.org/cn/index.php/Cool_Solution-%20LDAP_search_user)

### Behaviour settings

These settings control the behaviour of the connector.

```json
{
  ...
  "behaviour": {
    "write_changes_to_opsi": true,
    "root_dir_in_opsi": "clientdirectory",
    "update_existing_clients": true,
    "prefer_location_from_directory": true,
    "group_handling": "dn",
    "group_description": "dn",
    "override_root_dir": true
  },
  ...
}
```

If `write_changes_to_opsi` is set to `false`, no data will be written to opsi. This setting can be used to check the settings before they are applied.

`root_dir_in_opsi` specifies which group should be used as the root group in opsi. You have to ensure that this group exists.

The group `clientdirectory` is shown in configed as `DIRECTORY`. So if clients or groups should appear directly in `DIRECTORY`, the value `clientdirectory` must be entered as the value for `root_dir_in_opsi`.

If `update_existing_clients` is set to `false`, clients already existing in opsi will not be changed. If this is set to `true`, any manually entered data will be overwritten with the values from the directory.
If `prefer_location_from_directory` is set to `true`, clients in opsi are moved to the position they have in the directory. To deactivate this behavior, this value must be set to `false`.

Since version 31, group handling can be controlled via the optional key `group_handling`. The default is `cn`. Groups are derived from the DN of a computer and created accordingly as part of the opsi directory. A client can only be member of one group.

If the group handling is set to `ucsatschool`, the behavior is adapted for use in UCS@School environments. The opsi-directory-connector will automatically search for schools and find their rooms, which are then synchronized to opsi. For every school a group will be created in opsi. In UCS@School a computer can be a member of multiple rooms. To follow this behaviour, the groups are not created as a group within the opsi directory, but as a normal group, so that a client can also be found in multiple groups.

If the groups in OPSI should be created under `DIRECTORY` after all, the switch `override_root_dir` can be set to `false`. `override_root_dir` is only available for `group_handling uscatschool` and the default value is `true`. If `override_root_dir` is set to `false` and the groups are thus stored in OPSI under `DIRECTORY`, it should be ensured that each school computer has only been assigned to one room.

With `group_description` the description of the OPSI groups can be customized. The following values are possible:

- `dn`: The dn of the group is stored in OPSI as the group description.
- `directory`: The group description is read from the `description` field of the directory group.
- If the value is not set or is set differently, the description is filled with the name of the group.

```
... 
"behaviour": {
   "group_handling": "ucsatschool",
   ...
   "group_not_in_directory": true,
   "opsi_clients_to_ignore": {
      "clients": ["win1.uib.local","win2.uib.local","win3.uib.local"],
      "groups": ["server"]
   }
}
... 
```

If `group_not_in_directory` is set to `true`, all clients that are not in the directory are added to the `not_in_directory` group. This option is only available with `group_handling uscatschool`. With the `opsi_clients_to_ignore` parameter clients or whole groups can be excluded from this rule.

**Mappings**

With a system as flexible as a directory service, the connector needs information about which attributes in the directory should be matched to which attributes in opsi.
There is a mapping for client attributes. The key of the mapping is the attribute in opsi and the value is the attribute from the directory service. If the value (in the mapping) is empty, no mapping will be done.

If the value read from the directory for the ID of the client is not recognizable as a FQDN, a corresponding FQDN will be created. The domain part for this is formed from the DC values of the element.

On Univention Corporate Server (UCS), the value for `hardwareAddress` can be set to `macAddress` if the connection is established via LDAP (port 7389 or 7636).

In the "mapping" area, the assignment of clients to depots can be defined. Currently there is only the mapping type "network". With the "network" type, a client is assigned to a depot if the ip address of the client matches the network address range ("networkAddress") of the depot. Alternatively, a list of network ranges can be assigned to a depot.
**Manual assignment of group names**

Group names can usually be used without any major adjustments. However, it can happen that group names exist which are invalid in opsi.

For these special cases, a manual assignment of group names can be made, which allows these cases to be handled.

To configure this, an entry `group_name` is created in `mapping`, which holds the mapping from the directory service to opsi. The name is not changed for groups that do not appear in this assignment. Group names are always processed in lower case letters, which is why the entries here must be made in lower case letters. This is possible since version 23.

The following example maps the group `_server` originating from the directory service to the group `server` in opsi.

```json
{
    ...
    "mapping": {
        "client": {
            ...
        },
        "group_name": {
            "_server": "server"
        }
    },
    ...
}
```

Please be careful with this feature as it may introduce undesirable side effects. Therefore, this option should only be used for special cases.

**opsi connection settings**

This controls how the connector connects to opsi.
Enter the address of the opsi server under `address`. Do not forget to specify the port!

A proxy for the connection can be set via the environment variable `HTTPS_PROXY`.

User and password are used for authentication to the opsi server. If a value is given for `passwordFile`, this is interpreted as the path to a file that contains the password. The content of this file will be used as the password. This means that the password does not have to be stored in plain text in the configuration file. This will override the value set for `password`, if the file can be read.

We recommend using a dedicated user account. The creation of additional users is described in the *Getting Started* manual.

If the parameter `exit_on_error` is set to `true`, then any problem when updating the data in opsi - this can also be triggered by submitting values that are invalid in opsi - results in a break. If this is set to `false`, errors are logged, but the execution is not terminated.

Under `connection_options` options for the connection to the opsi server can be set. The verification of the server certificate is controlled by means of `verify_certificate`. This value should be set to `false` when using self-signed certificates.

Since version 14 it is possible to test the connection to the opsi server via the parameter `--check-opsi`, without establishing a connection to the directory service.

**9.3.5. Running the connector**

After the installation a binary called `opsi-directory-connector` will be present on the system.

It is required to pass an argument `--config` together with the path to the configuration file.

```
opsi-directory-connector --config /etc/opsi/opsi-directory-connector-custom.conf
```
The user running the binary does not require any access to opsi as this is all stored in the configuration file.

**Example: recurring runs with systemd**

The connector currently does one synchronisation run when executed, but the chances are good that you want to have a constant synchronisation of data.

It's easy to automate the execution of repetitive runs.

We will be using systemd for this. In contrast to cronjobs, systemd will prevent simultaneous runs from occurring, which is why systemd is a good choice.

The following example will set up the connector to run five minutes after the machine starts, and every hour thereafter.

The following two files must be stored under `/etc/systemd/system/`, the directory for user-defined units. One file is the timer, which calls our job repeatedly and the other is for the job itself.

Please fill the file `opsi-directory-connector.timer` with the following content:

```ini
[Unit]
Description=Start the opsi-directory-connector in regular intervals

[Timer]
OnBootSec=5min
OnUnitActiveSec=1hour

[Install]
WantedBy=timers.target
```

And this is the content of `opsi-directory-connector.service`:

```ini
[Unit]
Description=Sync clients from AD to opsi.
Wants=network.target

[Service]
Type=oneshot
ExecStart=/usr/bin/opsidirectoryconnector --config /etc/opsi/opsidirectoryconnector-custom.conf
```

To enable the timer and start it right away, the following commands can be used:
systemctl enable opsi-directory-connector.timer
systemctl start opsi-directory-connector.timer

If the timer does not get started, it will start to run the next time the machine is restarted.

**Example: recurring runs as cronjob**

It is easy to automate the execution of recurring runs through a cronjob.

Please note that simultaneous runs can take place, which is why it is best to choose a larger synchronization interval. To avoid this problem, it is recommended to use **systemd** instead of **cron**!

The cronjob file can usually be edited through `crontab -e`. For a synchronisation that happens every hour the following can be used:

```
0 * * * * /usr/bin/opsidirectoryconnector --config /etc/opsi/opsidirectoryconnector-custom.conf
```

### command line arguments

**Usage:** `__main__.py [-h] [--version] [--log-level {0,1,2,3,4,5,6,7,8,9}] [--log-level-stderr {0,1,2,3,4,5,6,7,8,9}] [--log-level-file {0,1,2,3,4,5,6,7,8,9}] [--log-file LOG_FILE] [--max-log-size MAX_LOG_SIZE] [--keep-rotated-logs KEEP_ROTATED_LOGS] [--check-directory | --check-opsi | --delete-clients DELETE_CLIENTS ...] [--dry-run] --config CONFIG

If an arg is specified in more than one place, then commandline values override environment variables which override defaults.

**optional arguments:**
- `-h`, `--help`
  show this help message and exit
- `--version`
  show program's version number and exit
- `--log-level {0,1,2,3,4,5,6,7,8,9}`
  Sets how much information will be logged. [env var: `OPDC_LOG_LEVEL`]
  (default: 4)
- `--log-level-stderr {0,1,2,3,4,5,6,7,8,9}, -l {0,1,2,3,4,5,6,7,8,9}`
  Sets how much information will be logged. [env var:
--log-level-file \{0,1,2,3,4,5,6,7,8,9\}
Sets how much information will be logged to the log file.

[env var: ODC_LOG_LEVEL_FILE] (default: 5)
--log-file LOG_FILE

--max-log-size MAX_LOG_SIZE
Limit the size of logfiles to SIZE megabytes. Setting this to 0 will disable any limiting. [env var: ODC_MAX_LOG_SIZE] (default: 5.0)

--keep-rotated-logs KEEP_ROTATED_LOGS
Number of rotated log files to keep. [env var: ODC_KEEP_ROTATED_LOGS] (default: 1)

--check-directory
Check if a connection to the directory can be established and if items will be received. (default: False)

--check-opsi
Check if a connection to the opsi server can be established. (default: False)

--delete-clients DELETE_CLIENTS [DELETE_CLIENTS ...]
Delete list of clients from directory. (default: None)

--dry-run
Print what would be done. (default: False)

--config CONFIG
Path to the config. (default: None)

Starting with version 39 the opsi-directory-connector uses the OPSI logger with loglevel 0-9. By default logs are written to /var/log/opsi-directory-connector and to stderr. With the parameters --log-level-stderr, --log-level-file the log level of the respective log can be set. --log-file defines the log file.

The log file is rotated after a specified size (default 5 MB) and by default one of these rotated log files is kept. The parameters --max-log-size and --keep-rotated-logs can be used to override the default values.

In addition to the command line parameters, the values can also be set in the configuration file or via environment variables. The following applies here:

- Parameters overwrite everything
- Environment variables overwrite configuration and default
- Configuration overwrites default
Example configuration:

```json
{
  ...
  "log-level-stderr": 6,
  "log-level-file": 3,
  "keep-rotated-logs": 4
  ...
}
```

If the directory connector is started with the `--dry-run` option, the individual steps are output without making any changes in OPSI. With `dry-run`, in contrast to the configuration parameter `write_changes_to_opsi: false`, the output is adjusted, to give a better overview of the actions. Example output:

```
---------- opsi actions ----------
Creating client client1.opsidc.intranet.
Creating client ds-win-client-2.opsidc.intranet.
Creating client ds-win-client-1.opsidc.intranet.
Creating client mac-client-1.opsidc.intranet.
Creating client windows-client-1.opsidc.intranet.
Creating client raspberrypi-1.opsidc.intranet.
Adding mac-client-1.opsidc.intranet to opsitestschool-mac pool.
Adding windows-client-1.opsidc.intranet to opsitestschool-pc pool og1.
Adding ds-win-client-2.opsidc.intranet to depotschule-pool-1.
Adding ds-win-client-1.opsidc.intranet to depotschule-pool-1.

---------- summary ---------------
Create 6 clients and 0 groups.
0 clients removed from group.
Adding 4 clients to a new group.
```

### 9.3.6. Delete Clients from the directory

In the standard case the opsi-directory-connector has only read access to the directory. With the start parameter `--delete-clients` an independent run is started, which tries to delete the given objects from the directory. For example:

```
opsi-directory-connector --config config.conf --delete-clients client1
```

The objects can be specified more precisely:
opsi-directory-connector --config config.conf --delete-clients computers/test-clients/client1

This call would fit on object `cn=client1,ou=test-clients,ou=computers,dc=example,dc=org`, but not `cn=client1,ou=clients,ou=computers,dc=example,dc=org`. So the first object will be deleted.

Multiple clients can also be specified:

```
opsi-directory-connector --config config.conf --delete-clients computers/clients/client1 client2 client3
```

- The `delete-clients` function should be used with caution.
- The `dry-run` function can also be used with `delete-clients`.

9.4. opsi WIM Capture

9.4.1. Prerequisites for the opsi extension opsi wim capture

This module is currently a co-funded opsi extension. Some preconditions are required, in order to use this module. That means that you need a suitable modules file to unlock this extension. You can get this file by purchasing the extension module. For evaluation purposes you can get a temporary modules file without charge. (→ mail us at info@uib.de).

Technical requirements are opsi 4.0.6 with package versions:

<table>
<thead>
<tr>
<th>Table 5. required packages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>opsi-package</strong></td>
</tr>
<tr>
<td>opsi-linux-bootimage</td>
</tr>
<tr>
<td>opsi-client-agent</td>
</tr>
<tr>
<td>Windows Netboot &gt;=7</td>
</tr>
<tr>
<td>opsi-clonezilla</td>
</tr>
</tbody>
</table>

For the product `opsi-wim-capture` the share `opsi_depot_rw` must have read/write permission for the `pcpatch`. Check your Samba configuration.

- Since opsi-wim-capture Version 4.1.x there is a full uefi support
- install.esd (instead of install.wim) as target format is supported since opsi-wim-capture Version 4.1.x.
9.4.2. Quick Info

For the people who are looking for a quick guide, this is the place to start. More detailed information follows below.

Pre-requirements

- Set boot priority of PXE-Boot / LAN-Boot on Computer BIOS to first priority
- Computer must be configure with the following specifications on the properties of the Netboot-Products:
  
    boot_partition_size=0
    preserve_winpe_partition = true
    windows_partition_label = WINDOWS
    windows_partition_size=100%

- Target product completion:
  The target product used is usually one of the provided capture products, e.g.: win7-x64-capture
  Winpe and Drivers directory can be created as symbolic links from the standard product.
  The install files folder must be copied because the install.wim must be adapted.
  In addition, Files from the custom directory are copied or linked, e.g.: unattend.xml

- The property opsi-clonezilla imageshare = auto (is the default, deprecated //<servername>/opsi_images)
  imagefile and runcommand will be replaced automatically opsi-wim-capture

- All software, which should be integrated in the image, must be install on the computer.

- Start of the product opsi-wim-capture:*

- Completion from the following properties is required:

  image_description = <Image description>
  imagename = <image name>
  target_product = win7-x64-captured

- Set opsi-wim-capture on setup

Install the computer with the new image:

- Target product (e.g. win7-x64-captured) adjusting the following to setup:
  Imagename= (Apply the same name from the property opsi-wim-capture)

9.4.3. Introduction

With NT6 (i.e. from Vista), Microsoft has introduced a new Image Format/ Container for OS Installation. Its called **Windows Imaging Format (WIM)**. A WIM Image is not exactly an image of a disk or partition, but rather an archive of files and Metadata. A WIM file can contain several images. The default installation of an NT6 operating system works like this: setup.exe unzips an image from an install.wim file, configures it, and adds some additional drivers.

This way, an installation is quicker than with NT5. But then installation of Hotfixes with NT6 takes
significantly longer, so that the basic install of Windows 7 will take 30 minutes, but adding all required Hotfixes may take several hours.

With this opsi extension, it is possible to read an installed Windows OS, including installed software, Hotfixes and configurations, and save it as a WIM image. This WIM file can then be used as a source for a fresh Windows Installation.

Our product opsi-wim-capture is made exactly for this purpose. Roughly, we boot off a PE partition, so that the PE can read the system partition and save it into a WIM file.

### 9.4.4. Overview of the Sequence

Capturing an installed Windows image works like this:

**Preparation:**

- **opsi** - installation of a Windows OS with set property:
  
  ```
  preserve_winpe_partition=true
  boot_partition_size=0
  windows_partition_size=100%
  ```

- run the product **opsi-wim-capture**.

All the following steps will be controlled by the product **opsi-wim-capture**:

1. via opsi-clonezilla, makes a backup of the disk (OS and winpe partition)
2. backup of the opsi metadata
3. make winpe partition bootable, create winpe script (work.cmd)
4. sysprep of the installed systems (depersonalization)
5. boot winpe, capture of OS disk, write to destination product
6. restore of the original disk (OS and winpe partition) via opsi-clonezilla

### 9.4.5. Sequence Details

**Preparation**

Installation of a Windows OS must have the property set to **true** like this **preserve_winpe_partition=true**, because the winpe partition will be needed later.
After the Windows OS installation you can install additional Software and Hotfixes, configure the system manually or via opsi.

In case the property `disabled` is set to `true` (default=false), the process will be canceled immediately. This switch is for development only.

The setting of the property `always_backup_before_sysprep` will be checked. If yes, it’ll make a backup of the system via opsi-clonezilla.
In **opsi-clonezilla**, the `runcommand` is 

```
ocs-sr -q2 --batch -j2 -rm-win-swap-hib -i
2000 -p true savedisk imagefile sda
```

Within this command, `imagefile` will be set according to the value of the property `clonezilla_imagefile`. In case its set to `auto` (default), we’ll configure the value for `imagefile` automatically. This will be done with the help of property values and the client name according to the following pattern:

```
<FQDN of client>_<target_product>_<imagename>
```

If the value is not `auto`, the value contained will be used as `imagefile`. Furthermore, we’ll set the product opsi-clonezilla to setup. In order to initiate opsi-clonezilla, reboot.

In order to avoid a never ending loop, we write a reboot flag, so that after writing the backup, we can see that this step has already been done.

Technical note: We do not want to reboot again after restoring the backup, though (but the reboot flag is contained in the backup). That’s why the reboot flag is being set as a time stamp. In case the time stamp is older than 0.1 days (i.e. 2.4 hrs), it will be ignored.

The system will reboot now, leaving the product opsi-wim-capture set to `setup`. opsi-clonezilla will start up and do the backup.

---

### Why backup via opsi-clonezilla?

The sysprep action to follow will leave the OS partition unusable. An OS, that is set up from a captured WIM Image, will contain information about the sysprep run. Thus, it cannot be used for further capturing via opsi-wim-capture.

Only perform repeat capturing using a previously restored opsi-clonezilla image.

The product opsi-clonezilla is now being configured in a way that it will perform a restore on next run.
Now information about the installed opsi-products (and versions) will be stored on the client.

The productOnClient objects of all Localboot Products are being written to `c:\opsi.org\tmp\productonclients.json`.

The machine’s opsi-client-agent is now being deactivated, so that it cannot run after deployment based on this image.

In order to be able to deploy the captured image like a default Windows Setup to any machine, it needs to be depersonalized. This will be done via `sysprep`.
This does not mean all of the software will be depersonalized. It's, that installed software holds data regarding on which computer it was originally installed. A config of that kind will be likely to cause problems, specially if you deploy the image to different machines. It might be a good idea not to capture all the software on the computer.

If the property startcapture is set to false (default=true), will stop working after the sysprep, and shut down the machine. This makes sense only if you plan to take an image of the machine using a different tool.

Figure 83. Schema: Activating and boot-enable the PE partition

In order to read the OS partition and writing it into the WIM file, we have to use a Windows OS, which cannot be the Windows OS we want to read (for obvious reasons). Therefore, we use the Windows PE we created and preserved at the initial installation. And afterwards:

- Activation of the WinPE as bootable partition, creation of the required boot record, (if necessary) deactivation of drive letters of other partitions
- Reading opsi metadata concerning installed products on the client, saving the data to a temporary folder on the client
- some cleaning on the system we want to capture

Figure 84. Schema: Creating work.cmd inside the PE

- Writing a command file, which will initiate the capturing at next WinPE boot.
- Provisioning of further data for the WinPE run, like list of products from the property start_after_capture
- Reboot the client
Now the WinPE starts and will do the actual capturing. Here are the details:

- Mounting the `opsi_depot_rw` share, so that we can write to it.
- Checking the architecture of the WinPE (32/64 Bit); start of the corresponding opsi-script interpreter.
- Establishing a connection to the opsi-webservice
- Re-activation of the drive letters
- If the property `check_disk_before_capture` contains the value `true` (default=false), we perform `chkdsk` on the Windows OS partition. That will take time.
- Checking for existence of the target product stated in the property `target_product` on our `opsi_depot_rw` share, and whether it contains an `install.wim` file in the right place.
- Checking and creating a lock file within the `target_product` folder. If this file exists already, we cancel the process in order to prevent several capturing processes writing to the same WIM file.
- If the property `force_imagex` is set to `true` (default=true), then we use the `imagex` command of our product `opsi-wim-capture` for capturing, even if the Windows PE has a `dism` command. Otherwise `dism` will be used, if available. `Dism` is faster, but might produce images that cannot be used for successful deployment.
- If the property `capture_mode` is set to `append`: Check, if there is an image of that name contained in the `install.wim`, and delete it. The value `always_create` will only be accepted, if `dism` is being used. In this case, a new `install.wim` file will be created.
- Start of the capture process. The previously mentioned tool (`imagex vs dism`) and the `capture_mode` chosen will be used. The name of the image is set by the property `imagename`. The property `image_description` will determine the description of the image. This can take a long time to be completed.

Keep in mind the name of the Image! The name of the image created can’t be automatically added to the list of installable images at the current time. You have to keep the name in mind and state the image name when deploying!

- Deleting of the lock file in the `target_product` folder.
- Merging the resulting log files.
• Request action setup for products contained in the property setup_after_capture. At this time, also product dependencies will be resolved. The property is a list and can contain several product IDs.

leave settings so that opsi-clonezilla will be set to setup!
The machine will be depersonalized after the capture run, and thus remain unusable. Our product opsi-clonezilla is prepared, so that the backup taken earlier will be restored automatically.

• Deactivation of the WinPE partition, and re-activation of the OS partition (Windows).
• Transfer of the logfile to the server. It will be appended to the existing logfile of the opsi-wim-capture run.
• Reboot

If the product opsi-clonezilla has been set to setup, a restore of the disk is being performed automatically.

![Diagram](image)

**Figure 86. Schema: Restore using opsi-clonezilla**

### 9.4.6. Products

**Main Product opsi-wim-capture**

The product opsi-wim-capture contains the following product properties:

• **always_backup_before_sysprep:**
  (true/false), Default=true,
  Always make an opsi-clonezilla backup before sysprep.

• **startcapture:**
  (true/false), Default=true,
  Sets the product opsi-local-image-capture to setup, and reboots the machine

• **disabled:**
  (true/false), Default=false,
  If set to true, nothing will happen. Its only there for debugging.

• **target_product:**
  Name of the target product: (Default = ”)
This property is not *intelligent*, i.e. we do not check, if the image being copied matches the target product. You could easily write a win7-32Bit Image into a Win81-64Bit product without errors. But you should not do that! Furthermore, we recommend separated products for capturing, that are only target products in the capture process. (for instance *win7-x64-captured*).

The target product has to be prepared for deployment like any other Windows OS product. The target file within the target product will be the *install.wim* file (**installfiles/sources/install.wim**), which also contains the images provided by Microsoft. Our new OS image will either be created as a new *install.wim*, or be appended to the existing file. This is being controlled by the property :

- **capture_mode**: (append/always_create) Default=*append*:

  - **append** will append the newly created image to the existing install.wim.

  - **always_create** will not work with a WinPE based on Windows < 8.

An *install.wim* file is a container, that can contain several images. All of them have a name and a description, which can be controlled by the following properties:

- **imagename**: Default = ''

- **image_description**: Default = ''

- The property **start_after_capture**

  this is a list of products that will be set to setup after completion of the capture process. Could be good to use for instance opsi-clonezilla, which will restore the backup taken before sysprep.

- **force_imagex**: true/false (default=true) This will use the imagex tool, even if dism is available.

- **opsi_depot_rw_host**: Normally auto (default) or leave empty.

  If not auto or empty: the host from which we mount the opsi_depot_rw share.

  If the host is given, it must be a hostname, a FQDN or a IP Number.

  This property is only for situations where the opsi_depot_rw share is not reachable at the opsi depot where the client is assigned to.

- **checkdisk_before_capture**: Should we make a file system check on the system partition before we capture.

  Default = false.

- **verify_clonezilla_images**: Should Clonezilla check the images after_save, before_restore, never, always
A verify check takes nearly the same time as the save or restore process. 
Default = never

Target Products

The target products shall host the captured images.

Why target products?

The target products do not differ from default opsi Windows netboot products. Technically, a standard win7-x64 they can be used as a target product.
We recommend the usage of dedicated target products, so that it is easy to differentiate a modified installation (opsi-wim-capture) from an unmodified installation (original Microsoft DVD).
Furthermore, it might be wise to keep a fall back product you can switch to in case the captured install.wim becomes unusable for some reason. This decision is for you to make.

We provide the following target products:

- win7-x64-captured
- win81-x64-captured
- win10-x64-captured

These products have to be equipped with required files from the OS DVDs like any other Windows netboot product (see our opsi-getting-started document).

Still, folders like winpe or drivers/drivers/additional/byAudit can be symbolic links to a directory in a suitable non-target product. Watch out: the Subfolder installfiles needs to be physically there (copy from Windows DVD).

9.4.7. Windows Installation via Target Product

(Deployment from a captured Image)

Restore of the opsi metadata from installed Products

The Problem:

If you reinstall a Windows with opsi, e.g. win7-x64, then during the installation of the opsi-client-agent all the local Boot products, which in this computer were previously marked as `installed`, will automatically be set to setup and thus reinstalled later.
This can not be completely carried out exactly in the rolling of a captured Image.
In the image is the backup from the opsi data that was stored during the capture process. This will be discovered when you install the opsi-client-agent and re-imported into the depot server. With it the products that were installed in the captured Image, now are on the newly installed computer mark as installed. Should now all the products that are mark as installed set to `setup`, this would imply that all products installed already in the image will be re-installed. This is not desirable.

By the restoring from the opsi metadata of installed products there are two alternatives available now
with opsi 4.0.7:

- **Alternative 1:**
  Restoring the metadata and retention of setup -Action Requests. Products that are mark as *installed* will **not** be set to *setup.* This is the default, and the behavior before opsi 4.0.7

- **Alternative 2:**
  Restoring the metadata. Products that are mark as *installed* will be set to *setup* except those which were contained in the restore metadata.

**Alternative 1**

By the deploy from a *captured* image, after the install, only the products which were already from the beginning of the OS-install set to *setup* will be automatically installed. These can be done through your intervention, or through the property *setup_after_install.* Therefore only the products which stood at *setup* before installing the operating system will be installed in this case.

This is the default, and the behavior before opsi 4.0.7

**Alternative 2**

Variant 2 behaves similar to what would be the case of an installation without a captured Image:

* Restore of the metadata.
* Products that are mark as *installed* are then set to *setup* except those which were contained in the restore metadata.

This behavior is only available since opsi 4.0.7 and is not the default. Option 2 is made possible by enhancements to the opsi script and is part of the opsi-client-agent of 4.0.7.

In order to be able to apply this behavior a *config* must be set on (*Host parameters*):

The *Boolean* configuration entry: `clientconfig.capture.switch_installed_products_to_setup`. If the entry for this client has the value *true* then variant 2 is applied, otherwise variant 1

*host parameter* can have specific client events activated or deactivated. The *host parameter* can be applied using the *opsi-configed* or the *opsi-admin*.

To create the *host parameter* over the *opsi-admin* the following commands are to be executed on the 'opsi-config-server':

```bash
opsi-admin -d method config_createBool
clientconfig.capture.switch_installed_products_to_setup
"capture.switch_installed_products_to_setup" true
```

With that you set for all computers **Alternative 2**.

To create the *host parameter* over the *opsi-server* select there *Server Configuration / ClientConfig* / And on the right side with the right mouse button: Add *Boolean configuration entry.*

### 9.4.8. Helper product opsi-wim-info

The product *opsi-wim-info* is useful to gather information about the images that are stored inside a
install.wim. These information is written to the logfile. Properties:

- **target_produkt**
  
  ProductId of the product where the install.wim file is searched.

### 9.4.9. Known Restrictions and Problems

The following restrictions are known as of today (13.7.2018):

- none

### 9.5. opsi Linux Support

#### 9.5.1. Supported as opsi-client: Linux

*Table 6. Supported Linux OS as Client in opsi 4.1 and 4.0.7 / Unterstützte Linux-OS als Client in opsi 4.1 und 4.0.7*

<table>
<thead>
<tr>
<th>Distribution</th>
<th>OS-Installation</th>
<th>netboot products</th>
<th>client-agent</th>
<th>opsicliented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian 10 <em>Buster</em></td>
<td>✓</td>
<td>debian, debian10</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Debian 9 <em>Stretch</em></td>
<td>✓</td>
<td>debian, debian9</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Debian 8 <em>Jessie</em></td>
<td>✗</td>
<td>debian, debian8</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Debian 7 <em>Wheezy</em></td>
<td>✗</td>
<td>debian, debian7</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Debian 6 <em>Squeeze</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ubuntu Bionic 18.04 LTS</td>
<td>✓</td>
<td>ubuntu, ubuntu18-04</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ubuntu Xenial 16.04 LTS</td>
<td>✗</td>
<td>ubuntu, ubuntu16-04</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Ubuntu Trusty 14.04 LTS</td>
<td>✗</td>
<td>ubuntu, ubuntu14-04</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>RHEL 7</td>
<td>✓</td>
<td>rhel70</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>RHEL 6</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CentOS 7</td>
<td>✓</td>
<td>centos70</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CentOS 6</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLES 15 SP2</td>
<td>✓</td>
<td></td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SLES 15 SP1</td>
<td>✓</td>
<td></td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SLES 12 SP4</td>
<td>✓</td>
<td>sles12sp4</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SLES 12 SP3</td>
<td>✓</td>
<td>sles12sp3</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SLES 12 SP2</td>
<td>✓</td>
<td>sles12sp2</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>SLES 12 SP1</td>
<td>✓</td>
<td>sles12sp1</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Table 7: Linux netboot products and the used installer type in opsi 4.1 and 4.0.7 / Linux Netboot-Produkte nach Installer-Typ in opsi 4.1 und 4.0.7

<table>
<thead>
<tr>
<th>Netboot product</th>
<th>Installer</th>
<th>State</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>debian</td>
<td>opsi</td>
<td>✓</td>
<td>squeeze - buster</td>
</tr>
<tr>
<td>debian10</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>debian9</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>debian8</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>debian7</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu</td>
<td>opsi</td>
<td>✓</td>
<td>trusty - focal</td>
</tr>
<tr>
<td>ubuntu20-04</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu18-04</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu16-04</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu15-10</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu15-04</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu14-04</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>centos70</td>
<td>distribution</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
9.5.2. Preconditions for using the opsi Linux Support

Technical precondition is opsi 4.0.5 with following packet versions:

Table 8. Required packets

<table>
<thead>
<tr>
<th>opsi packet</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-linux-bootimage</td>
<td>&gt;= 20140805-1</td>
</tr>
</tbody>
</table>

The opsi support for Linux is based on a free Open Source component (the netboot products) and a co-funded component (the client-agent).

The opsi-linux-client-agent is a **co-funded opsi extension module**.

In order to use the opsi Linux extension module, an activation file is required, this file can be acquired by buying the extension module. To obtain a temporary activation file for evaluation, please email us at info@uib.de.

For further details on handling extension modules please refer to the opsi manual.
9.5.3. opsi-linux-client-agent: 15 Free starts

The opsi-linux-client-agent includes 15 Free starts by which the agent can be used without any activation.

In detail: After the initial Install from the opsi-linux-client-agent the opsi-script can be started 15 times in service context without the need of activation.
This gives you the possibility to set a Linux computer with the corresponding opsi-products for the configuration needed. For example, after the installation of the system you could use the product `l-opsi-server` to make of the newly installed computer an opsi-server.

For a long lasting maintenance of the installed Linux computers after the 15 free starts, it is recommended the activation of the feature, in order to continue taking advantage of its benefits.

9.5.4. Installation of the packages

The Linux-related packages can be loaded over the `opsi-package-updater`. By default it already has the repository for the linux packages activated.

You can use the following command to install the packages:

```
opsi-package-updater -v --repo uib_linux install
```

9.5.5. Introduction

A single management tool for Windows and Linux

The objective of the opsi Linux extension module is to provide an homogenous management system for heterogenous environments. The focus is on integrating both worlds into the same management processes and tools.

This means, that a Linux installation is triggered the same way as a Windows installation. The Linux opsi-client-agent is based on the same source code as the Windows client and provides (when applicable) the same opsiscript instruction sets.

Independent from Linux distribution

The opsi Linux Support is designed to be independent from any special Linux distribution. The following distributions are supported:

- Debian
- Ubuntu
- OpenSuse / SLES (Suse Linux Enterprise Server)
- RHEL (RedHat Enterprise Linux)
- CentOS
9.5.6. Linux netboot products v406 based on the distribution installer

With opsi v405 the installation of Linux targets has been controlled by the netboot product. The opsi v406 Linux netboot products are based on installer of the respective distribution.

This is a fundamental change of the structure and behavior of these products.

Overview of the changes:

• Like with the unattended Windows installation, the Linux installer is equipped with an answer file to configure the unattended installation.

• The installer of a distribution is not like with Windows an executable program, but is a combination of the distribution kernel and initrd implementation.

• The system installation including partitioning, LVM, and all the basic software, are performed by the installers and not by the bootimage anymore.

• For the Suse and RedHat like distributions, the installation sources have to be provided by you by introducing the DVD as an ISO-file on the depot share. This is comparable to the Windows installation, with the difference that the Windows installation files are stored in a different place and stored as the content of the DVD and not as an ISO file.

• For the Debian like distributions, the installation sources are taken from the internet. Just the netboot versions of the distribution kernel with the associated initrd are placed on the depot share. These files are small, so they are included in the opsi package.
  Since opsi 4.0.7 it is also possible to provide for some netboot products a local http repository.

• For further maintenance of the installation the opsi-linux-client-agent can be installed with the basic installation.

Description of the automated installation process:

1. The opsi-linux-bootimage boots, deletes the partition table and creates a small temporary partition.

2. The opsi-linux-bootimage fetches the initrd for the distribution and unpacks it on the temporary partition.

3. The opsi-linux-bootimage fetches the generic answer file, patches it (personalisation) and moves it to the initrd directory.

4. The opsi-linux-bootimage creates some helper scripts and configuration files (e.g. for installing the opsi-linux-client-agent) and moves them to the initrd directory.

5. The opsi-linux-bootimage packs the patched initrd directory

6. The opsi-linux-bootimage boots the distribution kernel with the patched initrd per kexec.

7. The distribution kernel with the patched initrd performs the unattended installation of the target system and finally installs the opsi-linux-client-agent.

Advantages:
• The installation is done as specified by the distributor, which is of special importance for providing support in the business context.

• The opsi integration of new releases is easier and faster available.

• For Suse and RedHat like distributions, the installation is done from the sources on the opsi-server, and therefore is faster and more stable than installing from the internet repositories.

Disadvantages:

• Currently there is no UEFI support available. The information about UEFI gets lost during kexec boot. We hope to fix this with a future bootimage.

Providing the installation media on the server

For Suse and RedHat like distributions, the installation media is provided by an additional nfs-share: opsi_nfs_share.

To configure this share, there must be a NFS server installed and configured on the opsi-server:

Since opsi v4.0.6 stable this is done by the special package opsi-linux-support. This package is not installed by default and must be installed manually once, e.g.

```
apt install opsi-linux-support
```

If a firewall is running on your system you need to configure it to accept TCP connections at port 80. Please consult the appropriate manual on how to do this.

The opsi-linux-support package performs the following tasks:

• Installation of the applicable nfs-server package on the opsi-server. For Debian, Ubuntu, Suse this is the package: nfs-kernel-server. For Centos, Redhat it is the package nfs-utils.

• The share opsi_nfs_share is created and exported:

  ◦ Create directory:
    
```
    mkdir -p /var/lib/opsi/depot/opsi_nfs_share
```

  ◦ Add the share entry to /etc/exports:
    
```
    /var/lib/opsi/depot/opsi_nfs_share *(ro,no_root_squash,insecure,async,subtree_check)
```

  ◦ Activate the export:
    
```
    exportfs -r
```

  ◦ Check the successful export:
    
```
    showmount -e localhost
```

    The output should be:

    Export list for localhost: + /var/lib/opsi/depot/opsi_nfs_share *

• The share opsi_nfs_share has the following directory structure:

```
opsi_nfs_share/<productId>/<arch>/<dvd>.iso
```
example:
# opsi_nfs_share/opensuse13-2/64/openSUSE-13.2-DVD-x86_64.iso

The installation file must have an extension `.iso`, the name of the file does not matter. If there are several `.iso` files in the same directory, it is not specified which one to use.

- Copy the installation DVD to the `opsi_nfs_share` and execute:
  ```
opsi-set-rights /var/lib/opsi/depot/opsi_nfs_share
  ```

  IMPORTANT: use the standard installation DVDs of the distribution. Modified installation DVDs might have a different structure and therefore cannot be used in this context.

- If for any reasons the directory `/var/lib/opsi/depot/opsi_nfs_share` cannot be exported by the opsi-server per NFS (for instance because the depot share is already a NAS NFS share), so the NFS share to be used can be specified by a server config. Like:
  ```
  clientconfig.opsi_nfs_share=172.16.166.1:/var/lib/opsi/depot/opsi_nfs_share
  ```

The opsi v406 netboot products for Debian and Ubuntu do not get their installation files from an ISO file. They are provided by opsi with the standard netboot kernel and initrd. All further packages are taken from the internet. To relieve the network connection, using an apt-cache might be useful.

See chapter Section 9.5.14, “Proxy for `.deb`-packages”

See chapter Section 9.5.6.5, “Setting up a local deb http repository”

---

**Start order of involved services for SLES 11**

It may occur that the `showmount` command exits with an error message like the following:

```diff
# showmount -e localhost
clnt_create: RPC: Program not registered
```

Please make sure that after installing the NFS-server a reboot has been made. Then the services `rpcbind` and `nfs-server` need to be started in that exact order.

The services can be restarted by the following commands:

```diff
# systemctl restart rpcbind.service
# systemctl restart nfs-server.service
```

Then `showmount` works as expected:

```diff
# showmount -e localhost
Export list for localhost:
/var/lib/opsi/depot/opsi_nfs_share *
```
Common properties of the opsi v406 Linux netboot products

The following properties are available with all v406 netboot products to configure the Linux installation:

- **askbeforeinst:**
  Starting an installation has to be confirmed from the client console? (Default=True)

- **architecture:**
  architecture selection, which affects the selection of bootimages and the installation architecture. (Default=64bit)

- **language or locale:**
  Which language/locale is to be installed. (Default=distribution dependent/de)

- **console_keymap:**
  keyboard layout to be installed. (Default=distribution dependent/de)

- **timezone:**
  Timezone to be installed? (Default=Europe/Berlin)

- **root_password:**
  root password. (Default=linux123)

- **user_password:**
  user password. (Default=linux123)

- **proxy:**
  Proxystring (if required) as: http://<ip>:<port>. (Default=)

- **install_opsi-client-agent:**
  Install the Linux opsi-client-agent (co-funded project: a module file is required for activation. (Default=True)

- **setup_after_install:**
  Which opsi products should be installed after the installation of the operating system (opsi products set to setup). (Default="")

The products debian7, debian8 and ubuntu14-04, ubuntu16-04

The basic OS installation files are taken from the internet.

This product has the productive state.

This product has the following additional properties:

- **online_repository:**
  distribution repository for the installation. (only for the Debian/Ubuntu family) (Default=distribution dependent)

- **encrypt_password:**
  Password for disk encryption (only if encrypt_logical_volumes=true)
  Example: linux123 Default: linux123
9. opsi Extensions

- **partition_disk:**
  Disk to use: first or complete device path Examples: "first", "/dev/sda", "/dev/sdb"
  Default: first

- **partition_method:**
  The method use for partitioning of the disk:
  regular: use the usual partition types for your architecture / lvm: use LVM to partition the disk / crypto: use LVM within an encrypted partition Possible: "regular", "lvm", "crypto"
  Default: lvm

- **partition_recipe:**
  The kind of partitions that will be used:
  atomic: all files in one partition / home: separate /home partition / multi: separate /home, /usr, /var, and /tmp partitions Possible: "atomic", "home", "multi"
  Default: atomic

- **desktop_package:**
  Default: standard

- **language_packs:**
  Default: de

Videos (time lapse)


The product ucs41 and ucs42

The basic OS installation files are taken from the the official UCS repositories.

This product has a productive state. With this product one can install a master, slave, backup and even a memberserver. It is recommended to use the l-opsi-server localboot product to make an opsi server out of the UCS machine. Memberserver are supported when an opsi server is installed through l-opsi-server. This localboot products makes special adjustments to the server, so it can deploy clients like other UCS roles.

This product has the same properties as described above for debianX or ubuntuX and the following additional ucs specific properties:

- **dns_domain:**
  The DNS domain name: Example: example.com Default: ucs.test

- **ldap_base:**
  The ldap base. Example: dc=example,dc=com Default: dc=ucs,dc=test
• **ucs_code_name:**
  The code name of the ucs version that is provided in the repository.
  Example: ucs414 Default: ucs414

• **organisation:**
  The name of the organisation that will be used for the ucs installation.
  Example: uib gmbh Default: uib gmbh

• **windomain:**
  The name of the Samba/Windows domain.
  Example: MYDOMAIN Default: MYDOMAIN

• **external_nameserver:**
  Which nameserver is included to the ucs installation ?
  Example: 10.11.12.13 Default: auto = the name server given by dhcp

• **ucs_master_ip:**
  What is the IP Number of the UCS Domain Controller (needed for other roles to join) ?
  Example: 10.10.10.10 Default: 10.10.10.10

• **ucs_master_admin_password:**
  What is the password of the user Administrator of the UCS Domain Controller (needed for other roles to join) ?
  Example: linux123 Default: linux123

• **ucs_role:**
  Which UCS role should be installed ?
  Possible: "domaincontroller_master", "domaincontroller_backup", "domaincontroller_slave", "memberserver", "base"
  Default: domaincontroller_master

**Setting up a local deb http repository**

With debian8, ubuntu 16-04 and ucs41 package it is now possible to install from a local Apache2 repository.

To do that on Product as property " you must introduce the Address based on this example http://<opsi-server>/opsi/<productId> e.G http://opsiserver/opsi/debian8

Furthermore, the local repository must be of course created.

To do this please make sure that the product opsi-linux-support is installed on your opsi-server. This package installs the required packages for each Distribution(apache2) and also creates the necessary folders. These must be then completed with the corresponding Distribution repository.

To that end there are two possibilities:

1. Simple: You download a built and tested Repository from us and unpack it
2. Elaborate: You build it yourself

Simple:
Execute the following command as root.
Note that the path to the Apache2 DocumentRoot has different defaults which are dependent to each distribution.
Therefore, you may need to use the second line of the script!

**debian8**

```
#!/bin/bash
DOCUMENTROOT=/var/www/html
URL=http://download.uib.de/opsi4.0/products/opsi-linux
FILE=debian8.tgz
mkdir -p ${DOCUMENTROOT}/opsi
cd ${DOCUMENTROOT}/opsi
wget ${URL}/${FILE}
tar xzf ${FILE}
opsi-set-rights .
```

**ubuntu16-04**

```
#!/bin/bash
DOCUMENTROOT=/var/www/html
URL=http://download.uib.de/opsi4.0/products/opsi-linux
FILE=ubuntu16-04.tgz
mkdir -p ${DOCUMENTROOT}/opsi
cd ${DOCUMENTROOT}/opsi
wget ${URL}/${FILE}
tar xzf ${FILE}
opsi-set-rights .
```

**ucs41**

```
#!/bin/bash
DOCUMENTROOT=/var/www/html
URL=http://download.uib.de/opsi4.0/products/opsi-linux/univention-repository/
FILE=univention-repository-4.1.tgz
mkdir -p ${DOCUMENTROOT}/opsi
cd ${DOCUMENTROOT}/opsi
wget ${URL}/${FILE}
tar xzf ${FILE}
opsi-set-rights .
```

**ucs42**

```
```
#!/bin/bash
DOCUMENTROOT=/var/www/html
URL=http://download.uib.de/opsi4.0/products/opsi-linux/univention-repository/
FILE=univention-repository-4.2.tgz
mkdir -p ${DOCUMENTROOT}/opsi

cd ${DOCUMENTROOT}/opsi
wget ${URL}/${FILE}
tar xzf ${FILE}
opsi-set-rights.

Please take notice of the following file:
http://download.uib.de/opsi4.0/products/opsi-linux/univention-repository/opsi-ucs-repository-readme.txt

Alternatively one can create an own repository from an UCS DVD.

An own repository created from an UCS 4.2-0 DVD leads to an uninstalable state. The package depootstrap on this DVD is not able to install the system without interaction. However our provided repository is not affected by this.
#!/bin/bash
set -x

BASE_DIR=/var/www/opsi
DVD_PATH=UCSISOMOUNTPOINT
UCS_VERSION=4.1
UCS_SUBVERSION=4
UCS_REPODIR=univention-repository/mirror
UCS_REPODIR2=${UCS_VERSION}/maintained/${UCS_VERSION}-${UCS_SUBVERSION}
UCS_RELEASE_PATH=dists/ucs414/main/binary-amd64/Release

cd ${BASE_DIR}
mkdir -p ${UCS_REPODIR}
cd ${UCS_REPODIR}
pwd
ln -s . univention-repository
mkdir -p ${UCS_REPODIR2}
cd ${UCS_REPODIR2}
pwd
cp -r ${DVD_PATH}/all .
cp -r ${DVD_PATH}/amd64 .
cp -r ${DVD_PATH}/dists .
mkdir -p i386
cd all
dpkg-scanpackages . /dev/null | gzip -9c > Packages.gz
dpkg-scanpackages . /dev/null > Packages.gz
cd ..
cd amd64
dpkg-scanpackages . /dev/null | gzip -9c > Packages.gz
dpkg-scanpackages . /dev/null > Packages.gz
cd ..
cd i386
dpkg-scanpackages . /dev/null | gzip -9c > Packages.gz
dpkg-scanpackages . /dev/null > Packages.gz
cd ..
echo "Archive: stable" > ${UCS_RELEASE_PATH}
echo "Origin: Univention" >> ${UCS_RELEASE_PATH}
echo "Label: Univention" >> ${UCS_RELEASE_PATH}
echo "Version: ${UCS_VERSION}.${UCS_SUBVERSION}" >> ${UCS_RELEASE_PATH}
echo "Component: main" >> ${UCS_RELEASE_PATH}
echo "Architecture: amd64" >> ${UCS_RELEASE_PATH}
cat ${UCS_RELEASE_PATH}
cd ${BASE_DIR}
chown -R www-data:www-data univention-repository
echo "all done"
The products sles11sp4, sles12, sles12sp1

This product has the following additional properties:

```plaintext
name: productkey
multivalue: False
editable: True
description: email:regcode-sles for suse_register. Is only used if the host parameter 'license-management.use' is set to false. If it set to True the license key will be get from the license management module. / La clé de licence pour l'installation. Est utilisée uniquement si dans "Réseau et paramètres supplémentaires" 'license-management.use' est défini à false (faux). Si c'est réglé sur True (vrai) la clé de licence sera obtenue du module de gestion des licences.
values: ["", "myemail@example.com:xxxxxxxxxxxxxxxx"]
default: [""]

name: suse_register
description: set to false, if you don't want to register your system online, if you set this to false you have to give local repositories
default: True

name: local_repositories
multivalue: True
editable: True
description: list of local repositories to use. Syntax: "repository description",
example entry: "http://sles.example.com/suse/repo NameForRepo"
values: [""
default: [""

name: install_unattended
description: If false then do interactive installation
default: True
```

Source of installation

To download the installation DVD you need an account on SUSE. Installation DVD should have the name (we've made tests with files with these names): sles11sp4: SLES-11-SP4-DVD-x86_64-GM-DVD1.iso sles12: SLE-12-Server-DVD-x86_64-GM-DVD1.iso sles12sp1: SLE-12-SP1-Server-DVD-x86_64-GM-DVD1.iso Copy the ISO-File to /var/lib/opsi/depot/opsi_nfs_share/opensusel42-1/64/ Please don't forget to execute opsi-set-rights.

Videos (time lapse)

The following video shows an installation.
It is made with one frame per second and because of that, the installation that you see it is much more faster than a normal installation.

The products redhat70 and centos70

This product has the following additional properties:

```plaintext
name: install_unattended
description: If false then do interactive installation
default: True

name: selinux_mode
multivalue: False
editable: False
description: In which mode should SELinux run?
values: ["enforcing", "permissive", "disabled"]
default: ["permissive"]

name: partition_method
multivalue: False
editable: False
values: ["plain", "lvm", "btrfs", "thinp"]
default: ["lvm"]

name: productkey
multivalue: False
editable: True
description: email: regcode for subscription_register. Is only used if the host parameter 'license-management.use' is set to false. If it set to True the license key will be get from the license management module. / La clé de licence pour l'installation. Est utilisée uniquement si dans 'Réseau et paramètres supplémentaires' 'license-management.use' est défini à false (faux). Si c'est réglé sur True (vrai) la clé de licence sera obtenue du module de gestion des licences.
values: ["", "myemail@example.com:xxxxxxxxxxxxxxxxxx"]
default: ["]

name: subscription_register
description: set to false, if you don't want to register your system online, you need to set this to false and you have to introduce your local repository
default: True
```

Source of installation CentOS

Download the installation DVD here, e.g. from [here](#). Copy the ISO-File to `/var/lib/opsi/depot/opsi_nfs_share/centos70/64/` Please don't forget to execute `opsi-set-rights`.

Source of installation RedHat

To download the installation DVD you need an account on RedHat. Installation DVD should be named
we've made tests with a file with these name: rhel-server-7.0-x86_64-dvd.iso Copy the ISO-File to /var/lib/opsi/depot/opsi_nfs_share/redhat70/64/
Please don't forget to execute opsi-set-rights.

Videos (time lapse)
The following video shows an installation.
It is made with one frame per second and because of that, the installation that you see it is much more faster than a normal installation.

- http://download.uib.de/press-infos/videos/opsi-linux/centos70_406_1fps.mp4
- http://download.uib.de/press-infos/videos/opsi-linux/redhat70_406_1fps.mp4

9.5.7. Linux v405 netboot products without distribution installer

Basic OS installation per netboot

To install Linux on a client, at the beginning the standard opsi-linux-bootimage boots per netboot. It is the same image as the one used for the Windows installation.
The bootimage automatically performs the partitioning and formatting of the hard disc (/ and swap).
Next the installation of the basic Linux Operating System is performed (including network and ssh, but without X11). The installation process itself is quite different for the individual distributions, but has in common, that the installation is performed directly from the original distribution packages.

The basic Linux installation can be extended with optional opsi packages, for instance to turn the system into an opsi-Server (a new depotserver for instance).
Also the opsi-client-agent for Linux can be installed, which enables the automated installation and configuration of further software packages.

The opsi-client-agent for Linux is available as a co-funded opsi extension module, the required opsi netboot products for Linux installation are available as free Open Source modules.

Because the base installation is done from the Standard opsi-linux-bootimage, there are some distribution dependent differences, that have to be installed and configured after the first reboot of the installed system. This is for example the SELinux installation of the RedHat like or the keyboard configuration of the Debian like systems. These after boot installations and patches are done by the standard localboot product l-os-postinst.

Common properties of the v405 Linux netboot products

The following properties for controlling the Linux installation are available with all netboot products:

- **askbeforeinst:**
  confirm start of the new installation on the client? (default=true)

- **architecture:**
  architecture selection - affects the selection of the bootimage and the installation architecture. (default=64bit)
• **system_partition_size:**
  size of the system partition - the size may be given as percent of the hard disk size or as absolute size (G=Gigabyte). If you choose another value than 100%, the remaining rest will be used as data_partition. (default=100%)

• **swap_partition_size:** +size of the swap partition. (default=2000M)

• **data_partition_create:**
  create a data partition if there is some space left. (true/false) (default=true)

• **data_partition_preserve:**
  preserve an existing data partition?
  **always** = cancel the installation in case the preservation of an existing partition with the label *data* is not possible with the given partition data.
  **if_possible** = an existing partition with the label *data* is preserved if possible according to the given partitioning parameters. Otherwise it will be deleted.
  **never** = a new partition table will be created. (default=never)

• **language:**
  language / locale to be installed (default=de)

• **console_keymap:**
  keyboard layout to be used (default = distribution dependent / de)

• **timezone:**
  time zone to be configured (default=Europe/Berlin)

• **root_password:**
  root password (default=linux123)

• **user_password:**
  user password (default=linux123)

• **install_opsi_server:**
  install opsi server packages (default=false)

• **online_repository:**
  repository to use for installation - repository of the Linux distribution to be used for installation (not for SLES) (default = distribution dependent)

• **opsi_online_repository:**
  repository for opsi-server installation - repository for the opsi-server packets (default = distribution dependent)

• **proxy:**
  proxystring (if required) as: http://<ip>:<port> (default="")

• **additional_packages:**
  additional packages to install. Packages names separated by blanks. (default="")

• **wget_and_execute:**
  fetch a file via wget and execute it - URL (http) of a file to be executed at the end of installation. (default="")

• **install_opsi-client-agent:**
install the Linux opsi-client-agent (cofunding project: has to be activated by the /etc/opsi/modules) (default=false)

- **release:**
  (Debian and Ubuntu only) which release of the distribution is to be installed? (default = distribution dependent)

- **setup_after_install:**
  opsi product(s) to be installed after the OS installation is done (opsi products to be set to setup) (default=l-os-postinst)

**Netboot products for Linux distributions**

**ubuntu**

The basic installation is performed per debootstrap directly from the network.

This product has the status *productive*.

It is UEFI/GPT compatible (tested for release=trusty).

For this product applicable opsi-server packets are available, that can be installed by setting `install_opsi_server=true`.

**debian**

The basic installation is performed per debootstrap directly from the network.

This product has the status *productive*.

It is UEFI/GPT compatible (tested for release=wheezy).

For this product applicable opsi-server packets are available, that can be installed by setting `install_opsi_server=true`.

**9.5.8. opsi-linux-client-agent**

The opsi-client-agent for Linux is part of the cofunding project *Linux Agent*, which is liable to pay costs.

The opsi-client-agent for Windows is based on two components:

1. the service `opsiclientd`
2. the action processor `opsi-script / opsi-script-nogui`

The opsi-client-agent for Linux is based on the Linux port of the Windows client agent.

The `opsiclientd` is not ported to all supported Linux distributions. If no `opsiclientd` is available, it is substituted by a direct `opsiscriptstarter` call.

A Linux `opsiclientd` is available for:
Debian 7 / 8
• Ubuntu 12.04 / 14.04 / 16.04
• openSuse 13.2 / 42.1
• SLES 12 / 12SP1
• UCS 4.0 / 4.1

If there is no opsiclientd, so it is replaced by the opsiscr iptstarter, which performs the following opsiclientd tasks at system start:

• connect to the opsi-server: check whether actions are to be performed
• mount the depot share
• start the action processor
• unmount the depot share
• transfer the logfile to the server

The Linux action processor is named opsisc ript and is built from the same sources as the Windows opsi-winst. So on Linux the same scripting syntax is available as on Windows. All common features, that are not Windows specific, are available, as there are e.g.:

• file handling
• string and stringlist functions
• executing external scripts and programs
• communication with the opsi-Server
• patching config files

Of course Windows specific features (like patching the Windows registry) are not available on Linux, but there are some additional Linux specific functions like e.g.:

• getLinuxDistroType
• getLinuxVersionMap

Logging of the opsisc ript ist available (like with the opsi-winst on Windows).

Linux opsisc ript is available as a graphical version for working with X-Windows and a noGUI version for systems without graphical user interface.

opsi-linux-client-agent: Installation: service_setup.sh

This method is the first choice for installations on a single computer. service_setup.sh can also be used for maintenance or repair of a client. For mass roll-out, see the chapter below.

1. login to the Linux client with root privileges
2. mount the shared directory on the opsi server at \\<opsiserver\\>\opsi Depot to any mount point
3. change to directory `opsi-linux-client-agent` at the mountpoint
4. start as this place the script `.service_setup.sh`

The script connects to the opsi-webservice in order to create the server side client information and to get the pckey. The connection requires the user/password combination registered in the config.ini. If the connection fails, a login window pops up (or commandline questions), where the user can fill in a URL, user, and password. The provided user has to be a member of the group `opsiadmin`.

After the installation, the client will be restarted


The `opsi-deploy-client-agent` installs the opsi-client-agent directly from the opsi-server to the clients.

Requirements for the opsi-server:

- The Python 2 package `paramiko` has to be installed. It is available as `python-paramiko` on most distributions and can be installed through the corresponding package manager.

Requirements for the clients are:

- ssh access as root or as a user that has the possibility to run sudo

When using `opsi-deploy-client-agent` 4.1.0.1 or older with a non-root user this has to have the possibility of executing `sudo` non-interactively.

The program creates the client on the server, then copies the installation files and the configuration information including the pckey to the client. After copying the necessary information, `opsi-deploy-client-agent` starts the installation on the client.

With `opsi-deploy-client-agent` a whole list of clients can be processed. These can include any number of clients that can be passed as the last parameter, or with the option `-f` by which the clients can be read from a file. When using a file, on each line a client must be present.

The program can work with IP addresses, host names and FQDNs. It will automatically try to recognize what kind of Address was passed.

The program can be found at `/var/lib/opsi/depot/opsi-linux-client-agent`

Run the program with root privileges.

It could happen that you must first make the program executable with:

```
Chmod u + x /var/lib/opsi/depot/opsi-linux-client-agent/opsi-deploy-client-agent
```

Please note that for the installation via `opsi-deploy-client-agent`, the values of ProductProperties (if the client is already registered) do not have any effect. This is because a connection to the backend can first be established during some part of the deployment. In the unusual case that a client is already registered and still the deploy program is used to roll out the opsi-linux-client-agent and non-default settings while desiring to set non-default settings, it is recommended to set opsi-linux-client-
agent to setup after successful application of the deploy program. This way the desired productProperties take effect and override the defaults.

```bash
bonifax:/var/lib/opsi/depot/opsi-linux-client-agent# ./opsi-deploy-client-agent --help
    [--debug-file DEBUGFILE] [--username USERNAME]
    [--password PASSWORD]
    [--use-fqdn | --use-hostname | --use-ip-address]
    [--ignore-failed-ping]
    [--reboot | --shutdown | --start-opsiclientd]
    [--hosts-from-file HOSTFILE]
    [--skip-existing-clients]
    [--threads MAXTHREADS]
    [--keep-client-on-failure | --remove-client-on-failure]
    [host [host ...]]

Deploy opsi client agent to the specified clients. The clients must
be accessible via SSH. The user must be allowed to use sudo non-interactive.

positional arguments:
  host                  The hosts to deploy the opsi-client-agent to.

optional arguments:
  -h, --help            show this help message and exit
  --version, -V         show program's version number and exit
  --verbose, -v         increase verbosity (can be used multiple times)
  --debug-file DEBUGFILE
                          Write debug output to given file.
  --username USERNAME, -u USERNAME
                          username for authentication (default: root). Example
                          for a domain account: -u "<DOMAIN>\<username>"
  --password PASSWORD, -p PASSWORD
                          password for authentication
  --use-fqdn, -c        Use FQDN to connect to client.
  --use-hostname        Use hostname to connect to client.
  --use-ip-address      Use IP address to connect to client.
  --ignore-failed-ping, -x
                          try installation even if ping fails
  --reboot, -r          reboot computer after installation
  --shutdown, -s        shutdown computer after installation
  --start-opsiclientd, -o
                          start opsiclientd service after installation
  --hosts-from-file HOSTFILE, -f HOSTFILE
                          File containing list of clients (one hostname per
                          line). If there is a space followed by text after the
                          hostname this will be used as client description for
                          new clients.
  --skip-existing-clients, -S
```

9. opsi Extensions

289 / 506
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--threads MAXTHREADS, -t MAXTHREADS</td>
<td>number of concurrent deployment threads</td>
</tr>
<tr>
<td>--keep-client-on-failure</td>
<td>If the client was created in opsi through this script it will not be removed in case of failure. (DEFAULT)</td>
</tr>
<tr>
<td>--remove-client-on-failure</td>
<td>If the client was created in opsi through this script it will be removed in case of failure.</td>
</tr>
</tbody>
</table>

**opsi-linux-client-agent: Installation: Via opsi netboot product**

If you install a Linux via opsi netboot product, you only have to switch the property `install_opsi-client-agent` to `true` (which is the default) in order to install the opsi-linux-client-agent.

**opsi-linux-client-agent: opsiclientd configuration**

The `opsiclientd` for Linux is a port of the `opsiclientd` for Windows. So it works with the similar configuration file which is located at: `/etc/opsi-client-agent/opsiclientd.conf`.

A detailed description of this file you will find at the chapter for the opsi-client-agent for Windows: Section 6.1.3.6, “Configuration”

At the moment there are not all of the features and events available on Linux. Available are:

- Start at boot time (or via explicit start of the service `opsiclientd`). At Linux is the name of this event `opsiclientd_start` (not `gui_startup`)
  - `event_on_demand`
  - The `event_timer` but only with the configuration: `super = default`

Not available are:

- Everything that is related to local caching (`WAN-Extension`).
- Modification of events via `preconditions`.
- The `opsiclientd` notifier
- The `event_net_connection`
- The `event_on_shutdown`
- The `event_silent_install`

**opsi-linux-client-agent: installation paths**

As usual on Linux, the linux-opsi-client-agent is spread to several directories:

The binaries:
9. opsi Extensions

/usr/bin/opsi-script (X11)

/usr/bin/opsi-script-nogui (without X11)

/usr/bin/opsiscriptstarter (preliminary opsiclientd replacement)

/usr/bin/opsiclientd

Auxiliary files:

Skin files:
/usr/share/opsi-client-agent/opsi-script/skin (deprecated)

Since opsi-script 4.12.0.31/opsi-linux-client-agent (4.1.0.9-1):
default: /usr/share/opsi-script/skin
custom: /usr/share/opsi-script/customskin

opsi-script Library:
/usr/share/opsi-script/lib

Translation files: /usr/share/locale/<LANG>/LC_MESSAGES/opsi-script.po

Config files:
/etc/opsi-client-agent/opsiclientd.conf (configuration of the opsiclientd报酬)

/etc/opsi-client-agent/opsi-script.conf (deprecated)
Since opsi-script 4.12.0.31/opsi-linux-client-agent (4.1.0.9-1):
/etc/opsi-script/opsi-script.conf

Logfiles / temporary files:
/var/log/opsi-client-agent

/var/log/opsi-client-agent/opsiclientd

/var/log/opsi-client-agent/opsi-script (deprecated)
Since opsi-script 4.12.0.31/opsi-linux-client-agent (4.1.0.9-1):
/var/log/opsi-script/

9.5.9. opsi-linux-client-agent: Known Bugs

Copy a bundle of files via Files section from a smb share may fail according to the Samba version This problem was reported from some samba3 Versions but seems to be vanished in samba4.

Workaround: Instead of:

```bash
[Files_copy_netboot]
copy -s "%scriptPath%/installfiles/*" "$target$/installfiles/
```
you may use:

```bash
[ShellInAnIcon_opsi_copy_netboot]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin

cd "$scriptPath%"
tar cf - installfiles | ( cd "$target$/installfiles/" ; tar xf - )
```

Script examples

For software deployment on Windows clients there can be said: the installation of software itself is as important as the subsequent configuring of the software. On Linux most packets are available from the distribution repositories. So the installation part is less, but the configuration part stays the same. Also there are applications, that are not available from the standard repositories. In this case special repositories or installation sources have to be added to the system. The important feature is, that all installation and configuration settings can be managed and logged on the opsi-server.

Here are some example snippets for an opsi-linux-client-agent opsi-script:

- exit in case the script detects a non Linux system
- detecting the distribution type (to use `apt`, `zypper` or `yum`)
- detecting the Linux version
- installing a packet
- adding a repository

Example: exit in case the script detects a non Linux system:

```ini
[Actions]
requiredWinstVersion >= "4.11.4.1"
ScriptErrorMessages=off

DefVar $OS$

set $OS$ = GetOS

if not($OS$ = "Linux")
    LogError "Wrong OS: Product: " + $ProductId$ + "is only for Linux"
    isFatalError "Wrong OS"
endif
```

Example: detecting the distribution type:
[Actions]
requiredWinstVersion >= "4.11.4.1"
ScriptErrorMessages=off

DefVar $distrotype$

set $distrotype$ = getLinuxDistroType

if $distrotype$ = 'debian'
    Message "Try to get Package Lock..."
    if waitForPackageLock("60","false")
        comment "we got the package lock."
    else
        LogError "could not get Package Lock"
        isFatalError "package lock failed"
    endif
    ShellInAnIcon_Upgrade_deb
else
    LogError "Wrong Distro: This Product is for Debian/Ubuntu only"
    isFatalError "Wrong distro"
endif

if not("0" = getLastExitCode)
    Message "failed ShellInAnIcon_Upgrade"
    LogError "failed ShellInAnIcon_Upgrade"
    isFatalError "failed Upgrade"
endif

[ShellInAnIcon_Upgrade_deb]
set -x
export DEBIAN_FRONTEND=noninteractive
apt update
apt --yes dist-upgrade
exit $? 

Example: detecting the Linux version and installing a packet:

[Actions]
requiredWinstVersion >= "4.11.4.1"
ScriptErrorMessages=off

DefVar $distCodeName$
DefVar $distroName$
DefVar $distRelease$
DefVar $desktop$
DefStringList $linuxInfo$

set $linuxInfo$ = getLinuxVersionMap
set $distCodeName$ = getValue("Codename", $linuxInfo$)
set $distRelease$ = getValue("Release", $linuxInfo$)
set $distroName$ = getValue("Distributor ID", $linuxInfo$)

set $desktop$ = GetProductProperty("desktop", "kde")

if $distrotype$ = 'suse'
    if $desktop$ = "unity"
        Message " No Unity on SUSE - fallback to KDE ..."
        set $desktop$ = "kde"
    endif ; unity

    Message "Try to get Package Lock..."
    if waitForPackageLock("60","false")
        comment "we got the package lock."
    else
        LogError "could not get Package Lock"
        isFatalError "package lock failed"
    endif
endif

if $desktop$ = "kde"
    if ($distroName$ = 'openSUSE project')
        ShellInAnIcon_kde_suse
    endif
    if ("SUSE LINUX" = $distroName$) and ($distRelease$ = "11")
        ShellInAnIcon_kde_sles11
    endif
    if not("0" = getLastExitCode)
        LogError "failed ShellInAnIcon"
        Message "failed kde"
        isFatalError "failed kde"
    endif
    endif ; kde
endif; suse type

[ShellInAnIcon_kde_suse]
set -x
zypper --no-gpg-checks --non-interactive install patterns-openSUSE-kde4 patterns-openSUSE-kde4_basis
zypper --no-gpg-checks --non-interactive install splashy-branding-openSUSE
exit $? 

[ShellInAnIcon_kde_sles11]
set -x
zypper --no-gpg-checks --non-interactive install --auto-agree-with-licenses -t pattern
Example: adding a repository:

```
[Actions]
requiredWinstVersion >= "4.11.4.1"
ScriptErrorMessages=off

DefVar $distCodeName$
DefVar $distroName$
DefVar $distRelease$
DefVar $desktop$

DefStringList $linuxInfo$

set $linuxInfo$ = getLinuxVersionMap
set $distCodeName$ = getValue("Codename", $linuxInfo$)
set $distRelease$ = getValue("Release", $linuxInfo$)
set $distroName$ = getValue("Distributor ID", $linuxInfo$)

set $desktop$ = GetProductProperty("desktop", "kde")

if $distroName$ = 'Ubuntu'
  if $desktop$ = "cinnamon"
    set $desktopPackage$ = $desktop$
    Message "Try to get Package Lock..."
    if waitForPackageLock("60","false")
      comment "we got the package lock."
    else
      LogError "could not get Package Lock"
      isFatalError "package lock failed"
    endif
  endif
else
  LogError "could not get Package Lock"
  isFatalError "package lock failed"
endif
ShellInAnIcon_ubuntu_cinnamon
if not("0" = getLastExitCode)
  Message "failed ShellInAnIcon_ubuntu_cinnamon"
  LogError "failed ShellInAnIcon_ubuntu_cinnamon"
  isFatalError "failed cinnamon"
endif
endif; ubuntu

[ShellInAnIcon_ubuntu_cinnamon]
set -x
export DEBIAN_FRONTEND=noninteractive
# we need to get the add-apt-repository command
```
apt --yes install python-software-properties
# the cinnamon repository
add-apt-repository ppa:gwendal-lebihan-dev/cinnamon-stable
apt update
apt --yes install ubuntu-desktop
exit $? 

9.5.10. Linux localboot products

Here some localboot products that are part of the standard opsi Linux support.

The product l-opsi-server

The product *l-opsi-server* serves to install on a Linux computer an opsi-server via opsi-linux-client-agent in an automated way. This can serve to install quickly a new opsi-depot-server or e.G. an opsi Test system.

Currently for a opsi-config server an other maschine can't be a opsi-linux-client and a opsi-depot-server at the same time.

To work around this limitation, you have two possibilities:

1. Using one opsi-config-server: After the installation of opsi via *l-opsi-server* and before you register this maschine as opsi-depot-server, you have to delete it as client in the configed.

2. Using two opsi-config-servers: Setup a second independent opsi-config-server, which is only used to administrate (install and mantain) your opsi-servers. So this second opsi-config-server knows the other opsi-servers only as linux-clients. Your other (first) opsi-config-server know theses other opsi-servers as depots.

In a UCS environment method 2 is recommended and the second opsi-config server must not be a UCS Server.

The product *l-opsi-server* has the following Properties:

- **opsi_online_repository:**
  (Base-) Repository for opsi-server installation.
  (Default="http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40")
  see also repo_kind

- **opsi_noproxy_online_repository:**
  (Base-) Repository for opsi-server installation (without any cache proxy).
  (Default="http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40")
  Should you require on opsi_online_repository to introduce a Proxy or deb-cacher (e.G. http://mydeb-cacher:9999/download.opensuse.org/repositories/home:/uibmz:/opsi/opsi40”), than introduce the URL without Proxy. Otherwise introduce the same as opsi_noproxy_online_repository.

- **repo_kind:**
  Which kind of repository ["experimental", "stable", "testing"] should be installed ?. (Default=*stable*)
With the Client OS, `opsi_online_repository` and `repo_kind` the URL will be built and the client will be added to an opsi repository.

- **backend:**
  Which backend should be installed? (mysql needs a valid activation file). (Default=`file`)
  A modules file with the require Activations can be stored in the custom directory of the product. If a modules file is found there, then will this one used.

- **opsi_admin_user_name:**
  The name of the opsi_admin_user to create (empty= nothing created). (Default=`adminuser`)
  If a user is introduced here, it will applied, also it will be added to the group `opsiadmin, pcpatch /opsifileadmin` and become as unix and samba password the value `opsi_admin_user_password`

- **opsi_admin_user_password:**
  What is the password of the opsi_admin_user to create (empty= not allowed). (Default=`linux123`)
  see `opsi_admin_user_name`

- **setup_after_install:**
  Which opsi product(s) should we switch to setup after l-opsi-server installation is done?.
  (Default=`""`)

- **allow_reboot:**
  May the server reboot if script is finished? (Default=`true`)

- **install_and_configure_dhcp:**
  Should we run the dhcp server on this machine? (Default=`False`)
  If this property is `false` then the following properties: `netmask`, `network`, `dnsdomain`, `nameserver` and `gateway` are meaningless because they are only used for dhcp configuration.

- **myipname:**
  Set a different IP name (FQDN) (`auto`= use standard) (Default=`auto`)
  Meaningless if `install_and_configure_dhcp=false`

- **myipnumber:**
  Set a different IP number (`auto`= use standard) (Default=`auto`)
  Meaningless if `install_and_configure_dhcp=false`

- **netmask:**
  Netmask (for dhcp). (Default=``255.255.0.0``)
  Meaningless if `install_and_configure_dhcp=false`

- **network:**
  network address (for dhcp). (Default=``192.168.0.0``)
  Meaningless if `install_and_configure_dhcp=false`

- **dnsdomain:**
  DNS domain (for dhcp). (Default=``uib.local``)
  Meaningless if `install_and_configure_dhcp=false`

- **nameserver:**
  Primary nameserver (for dhcp). (Default=``192.168.1.245``)
  Meaningless if `install_and_configure_dhcp=false`
• **gateway:**
  gateway (option routers for dhcp). (Default="192.168.1.245")
  Meaningless if `install_and_configure_dhcp=false`

• **ucs_master_admin_password:**
  Only needed for opsi installation on UCS Server with other Roles than Role Master.
  (Default=\texttt{linux123})

• **update_test:**
  Do not use: Internal Debugging. (Default=\texttt{False})

• **ucs_master_admin_password:**
  On a UCS machine the roles Slave, Backup and Member have to be joined correctly with the Master. This property takes the password to perform the join.

The product has *setup required before* dependency to the product \texttt{l-system-update}. That means when you set \texttt{l-opsi-server} on *setup* it will also automatically set \texttt{l-system-update} also on *setup* and installed before.

In the directory \texttt{custom} of the product \texttt{l-opsi-server} the activated file (\texttt{modules}) is stored, which is used in the Installation of the product \texttt{l-opsi-server} and will be preserved in the case of a new version of the product.

**l-os-postinst**

This product installs and configures those parts of the base installation, that cannot be done from the bootimage in a proper way.

This is for the different distributions:

• CentOS:
  - installation of SELinux

This product has a dependency to the product \texttt{l-system-update} which is executed before running \texttt{l-os-postinst}.

This product has a high priority, so it is executed before common products.

**l-desktop**

The product \texttt{l-desktop} installs a desktop packet on the computer.

The property \texttt{desktop} selects the desktop to be installed. Not all of the desktops are available for every distribution. For instance \texttt{Unity} is available for Ubuntu only. If the selected desktop is not available, the distribution specific default desktop will be installed. Furthermore the scope of the desktop packets differs according to the distribution and the selected desktop. It can be just the actual desktop software, or might also contain some base products like libreoffice, firefox, PDF Reader etc.

The property \texttt{desktop} can have the following values:
• Gnome
  Default for Debian, CentOS, RHEL.
  Available for all distributions.

• KDE
  Default for SLES, OpenSuse. Available for all distributions.

• Unity
  Available for Ubuntu only.

• Cinnamon
  Available for Ubuntu only.

• xfce4
  Available for Ubuntu, Debian.

• lxde
  Available for Ubuntu, Debian.

**l-system-update**

This product updates the system.

**l-swaudit**

Software inventory, based on the packet manager

**l-hwaudit**

Hardware inventory.
The hardware inventory currently is based on the Python implemented method as also used by the bootimage. Therefore the packet `python-opsi` from the opsi-repository of the distribution must be installed. So if there is no opsi-repository available for this distribution, the hardware inventory fails.

**l-jedit**

Java based editor with syntax highlighting for opsi-script. If Java is missing on the system, it will be installed automatically.

**9.5.11. Inventory**

To create an inventory, the data are collected on the client and sent to the server. The hardware inventory is based on the methods implemented in the bootimage.

The software inventory is based on the data from the packet management of the deployed Linux distribution.

**9.5.12. UEFI / GPT support**

Some of the opsi 4.0.5 Linux netboot products are UEFI/GPT compatible.
The 4.0.6 Linux netboot products are not UEFI/GPT compatible. At the moment they lose their information about the uefi environment at the kexec boot of the distribution kernel. We hope to fix this at a later point.

For details refer to the list of netboot products above or see at the UEFI chapter of the opsi-manual.

### 9.5.13. Roadmap

Linux support is a brand new opsi feature. Therefore not all of the planned features have been implemented yet with the first release.

Planned features to follow are:

- configurable partitioning
- logical volume management
- patching XML files
- patching hierarchical configuration files like dhcpd.conf

### 9.5.14. Proxy for .deb-packages

Instructions for installation and use of servers for local caching of debian packages:

- Ubuntu help: Apt-Cacher-Server
- How to set up Apt caching server on Ubuntu or Debian

### 9.6. opsi with UEFI / GPT

#### 9.6.1. Netboot products with UEFI support

**Table 9. opsi-clonezilla**

<table>
<thead>
<tr>
<th>Netboot product</th>
<th>Opsi 4.0.7 / 4.1</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-clonezilla</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

**Table 10. Standard Windows**

<table>
<thead>
<tr>
<th>Netboot product</th>
<th>Opsi 4.0.7 / 4.1</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server 2019</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Server 2016</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>win10 64 Bit</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>win10 32 Bit</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Server 2012 R12</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>win8.1 64 Bit</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Operating System</td>
<td>Status</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>win8.1 32 Bit</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>Server 2012</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>win7 64 Bit</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>win7 32 Bit</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>Server 2008 R2</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>winvista 32 Bit</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>winvista 64 Bit</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>Server 2008 64 Bit</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>winxp</td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>

✓: Supported ✗: Unsupported ◀: Under Development ◀: Discontinued

**Table 11. Linux**

<table>
<thead>
<tr>
<th>Netboot product</th>
<th>Opsi 4.0.7 / 4.1</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu18-04</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ubuntu16-04</td>
<td>✓</td>
<td>since ubuntu16-04_4.0.7.2-1</td>
</tr>
<tr>
<td>ubuntu14-04</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>debian</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>debian9</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>debian8</td>
<td>✓</td>
<td>since debian8_4.0.7.2-1</td>
</tr>
<tr>
<td>debian7</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>centos70</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>centos65</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>redhat70</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>opensusel15</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>opensusel15-1</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>opensusel42-3</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>opensusel42-2</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>opensusel42-1</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>sles15</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>sles12sp3</td>
<td>◀</td>
<td></td>
</tr>
<tr>
<td>sles12sp2</td>
<td>◀</td>
<td></td>
</tr>
</tbody>
</table>
Table 12. opsi-local-image

<table>
<thead>
<tr>
<th>Netboot product</th>
<th>Opsi 4.0.7 / 4.1</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-local-image-prepare</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>opsi-local-image-backup</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>opsi-local-image-restore</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>opsi-local-image-wim-capture</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>opsi-local-image-win*</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>opsi-local-image-ubuntu</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>opsi-local-image-opensuse13-2</td>
<td></td>
<td>▲</td>
</tr>
<tr>
<td>opsi-local-image-opensuse142-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opsi-vhd-win10-x64</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

✔: Supported ✗: Unsupported ✳: Under Development ▲: Discontinued

9.6.2. Netboot products with UEFI Secure Boot support

Table 13. Windows

<table>
<thead>
<tr>
<th>Netboot product</th>
<th>Opsi 4.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server 2019</td>
<td>✔</td>
</tr>
<tr>
<td>Server 2016</td>
<td>✔</td>
</tr>
<tr>
<td>win10 64 Bit</td>
<td>✔</td>
</tr>
<tr>
<td>Server 2012 R2</td>
<td>✔</td>
</tr>
<tr>
<td>win8.1 64 Bit</td>
<td>✔</td>
</tr>
</tbody>
</table>

✔: Supported ✗: Unsupported ✳: Under Development ▲: Discontinued

Table 14. Linux

<table>
<thead>
<tr>
<th>Netboot product</th>
<th>4.1</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu</td>
<td></td>
<td>✳</td>
</tr>
</tbody>
</table>
9.6.3. Preconditions for working with UEFI / GPT

This module currently is a co-funded opsi extension.
Some preconditions are required to work with that module, which is to get a suitable modules file to unlock the feature. You can get this unlock file by purchasing the extension module. For evaluation you can get a time limited modules unlock file without charge. (→ mail to info@uib.de).

Technical requirements are opsi 4.1 with package versions:

Table 16. required packages

<table>
<thead>
<tr>
<th>opsi package</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netboot products</td>
<td>&gt;=4.1</td>
</tr>
<tr>
<td>opsi server packages</td>
<td>&gt;=4.1</td>
</tr>
<tr>
<td>opsipxeconfd</td>
<td>&gt;=4.1.1.20-3</td>
</tr>
<tr>
<td>opsi-linux-bootimage</td>
<td>&gt;=20200506</td>
</tr>
</tbody>
</table>

9.6.4. Further remarks regarding the pxe-installation with the opsi-Moduls UEFI / GPT

- opsi supports only 64-Bit UEFI-Installations
- PXE-boot installs require a winpe that is capable of booting in UEFI mode. Often, an existing winpe will be capable of doing so, check by verifying there is a folder named EFI, as well as a file
named bootmgr.efi inside your winpe folder. If that is not the case, create a recent winpe as explained in our opsi-getting-started Manual, Chapter "Creating a PE". A winpe, that is UEFI capable, needs to reside in the winpe_uefi folder of the opsi netboot product. Provided your winpe is already capable of booting UEFI and MBR modes, you could simply place a softlink winpe_uefi > winpe.

• you have to configure your external DHCP server with the bootfile
  linux/pxelinux.cfg/shimx64.efi.signed

• Activate in the opsi-configed the checkbox "Uefi-Boot" for uefi-clients (since version 4.0.5.8.1) or set hostparameter clientconfig.dhcpd.filename=linux/pxelinux.cfg/shimx64.efi.signed This setting can also be made via the following command line call:

  opsi-admin -d method configState_create "clientconfig.dhcpd.filename" "here.host.id" "linux/pxelinux.cfg/shimx64.efi.signed"

If your opsi server had earlier versions of the package requirements installed you might need to edit the file /etc/opsi/opsipxeconfd.conf. The change involves the line uefi netboot config template x64 = /tftpboot/linux/pxelinux.cfg/install-elilo-x64. This line has to be changed to uefi netboot config template x64 = /tftpboot/linux/pxelinux.cfg/install-grub-x64 to ensure a compatibility between the UEFI bootloader and the named pipe.

• BIOS settings:
  Since the BIOS menus are very different and use different terminology, you need to consider here what is the best fit for your BIOS.
  ◦ Secure Boot disabled
    This entry is usually in the Boot or Startup but it can also be found in the Security area. Information about using secureboot can be found in a different chapter.
  ◦ Turn the BIOS on in the UEFI-Mode.
    If you have the choice between UEFI only, Legacy only or Both you should select UEFI only. If the selected option is Both this is not the best practice but it can eventually work. If the Legacy Support is present, it should be disabled. CSM Support in conjunction with UEFI only can remain enabled. Otherwise just disable it. UEFI Network Boot must be enabled. It could happened that the entry is also named Network Stack and also can be found under the UEFI category. If there are two categories for IPv4 and IPv6 here the right choice is IPv4.

9.6.5. Introduction

Recent PCs, tablets and server often are equipped with an UEFI BIOS. Often there is a legacy mode available to support the old features including PXE boot. But more and more devices come with an UEFI only BIOS (especially tablets). So they cannot be managed with the previous opsi environment.

To integrate these devices into opsi and to be able to use the advantages of UEFI, the uib gmbh developed the opsi extension for UEFI support.
9.6.6. What is UEFI and what is different about it?

UEFI is the abbreviation of *Unified Extensible Firmware Interface* and is the follow-up to the classic PC-BIOS (MBR-BIOS).

For detailed information on UEFI there are some links listed below.

UEFI has much more features than the old BIOS. Basically UEFI is a small operating system by itself. But in this place, we just consider some features, that are of special interes to the system administrator:

- The recent (by January 2014) implementations of UEFI by the hardware manufacturers have not developed any clear standards yet. As soon as the system is to be booted from any other device but the hard disc, you face the utter chaos. Often UEFI and classic BIOS are implemented both, sometimes they can be deactivated individually, or sometimes not. UEFI can be implemented with the Compatibility Support Module (CSM), or without. Netboot might work, or might not. Especially the availability of netboot is essential for structured client management.

- With the classic PC-BIOS the BIOS and its configuration usually are separated from the operating system. So BIOS configurations like the boot sequence cannot be changed by the operating system. This is different with UEFI. The operating system can change the boot sequence (and usually it does). This has consequences for a client management that relies on netboot.

- UEFI comes with its own boot manager, which not only can be used by the operating systems to change the boot sequence, but also contains the start entries for the operating systems themselves. This is to support the parallel installation of different operating systems, so that there is no conflict with the different boot loaders.

- UEFI can be implemented for 32 or 64 bit, which has to match a 32 or 64 bit operating system. A 32bit OS can not be installed on a 64 bit UEFI system.

- **Secureboot** allows only authorized systems to boot.

- partitioning with GPT and additional partitions for the bootloader:
  - 1. partition: EFI system partition (ESP) 100 - 260 MByte ; VFAT
  - 2. partition: Microsoft reserved (MSR) 32 - 128 MB; NTFS
  - following the actual OS partitions

Links:


9.6.7. What is different about GPT

GPT (GUID Partition Table) id the follow-up for the previous MBR partition tables. GPT is part of the UEFI specification.

The main features for the sysadmin are

- overriding the 2 Terabyte limit (now it is 8 Zebibyte)
Basic GPT can be used without UEFI. But UEFI depends on GPT. With UEFI there are up to two additional partitions:

1. the EFI system partition (ESP) with the bootloaders
2. Microsoft reserved (MSR)

Links:

http://de.wikipedia.org/wiki/GUID_Partition_Table

9.6.8. UEFI Boot

In contrary to the old BIOS the boot sequence not only can be defined for devices, but also can be set for different bootloaders on the EFI system partition. Furthermore the sequence can be changed by a running operating system. So if you set netboot as the first boot priority, this setting will not survive the first OS installation.

9.6.9. UEFI Netboot

Unfortunately early UEFI implementations do not support netboot at all, but netboot support is increasing.

With the UEFI support extension module uib gmbh has developed a succesfull UEFI netboot support for integrating UEFI clients into opsi. Because the UEFI standard is still under development and changing, in future the opsi UEFI module will continue to adapt to the technical changes, which might require structural redesigns of the module.

9.6.10. opsi support for UEFI netboot

The opsi support for UEFI is based on several components:

- adaption of the netboot UEFI bootloader GRUB2 to the opsi / client-management requirements.
- new opsipxeconfd, which also supports config files for the opsi-GRUB2 (in addition to the PXE config).
- new (64 bit) opsi-linux-bootimage with the tools for UEFI- and GPT management
- redesigned netboot products for OS installation (Windows/Linux) with additional support of UEFI/GPT (of course only for OS that support UEFI).
- client setting on the opsi-server whether to be treated as UEFI client or not.
• support of a software-controlled switch to UEFI netboot.

The label of the UEFI netboot entry of the firmware can be saved on the opsi-server (`clientconfig.uefinetbootlabel`), as far as the firmware supports it (there is an activatable netboot entry in the EFI). This allows opsi-products to use netboot for the next reboot. This technique is implemented in several opsi products. An important example is the product `opsi-uefi-netboot`:

This product tries to configure the UEFI for netboot and then triggers a reboot. If there is no `uefinetbootlabel` or the client is not an UEFI client only a reboot is triggered. This product is available for Windows and for Linux.

9.6.11. Installation

All packages required are installed automatically with opsi version 4.0.5.

9.6.12. Configuration of a Linux DHCP server

Configuration example of a Linux isc-dhcp-server:

```bash
filename "linux/pxelinux.0";

# this is the UEFI detection:
if substring (option vendor-class-identifier , 19,1 ) = "0" {
    log (info, "pxe client");
    filename "linux/pxelinux.0";
}
else if substring (option vendor-class-identifier , 19,1 ) = "6" {
    log (info, "efi32 client");
    filename "linux/pxelinux.cfg/elilo-x86.efi";
}
else if substring (option vendor-class-identifier , 19,1 ) = "7" {
    log (info, "efi64 client");
    filename "linux/pxelinux.cfg/shimx64.efi.signed";
}
else {
    log (info, concat ( "Unhandled vendor class Arch: ", substring (option vendor-class-identifier , 19,1 )));
}
```

See also: Fedora: PXE Boot Configuration

Example for the configuration of a Windows DHCP server 2012 R2

• As standard for this variant the PXE boot file for x64 Uefi installations is entered as default. DHCP options 66 and 67 are adjusted as it follows:

066 Host name of the start server: <IP of the opsi server>
067 Name of the start file: linux/pxelinux.cfg/shimx64.efi.signed
• To distinguish the BIOS clients, a manufacturer class identifier (PXEClient:Arch:00000:UNDI:002001) must be defined on the DHCP server:

Define manufacturer class
Add new manufacturer class
Edit class
  Display name: Legacy BIOS
  Ascii: PXEClient:Arch:00000:UNDI:002001

• The predefined options must be order under the manufacturer class:

Set predefined options
Options
  Option class: Legacy BIOS
  Add
Adjust the option type
  Name: Legacy BIOS
  Data Type: String
  Code: 60
  Description: PXEClient Class Legacy BIOS
Predefined options and values
  String: PXEClient

• Define the DHCP policy that is assigned to the boot file for the PXE boot (BIOS) of the manufacturer class:

New policy
  Policy Name: PXE BootFile Legacy BIOS
  continue
Add conditions
  Criteria: Manufacturer class
  Operator: equals
  Value: Legacy BIOS
  add
Would you like to configure an IP address range for the following policy: No
Manufacturer class: DHCP Standard Options
  067 Name of the start file
  file input
  String value: linux/pxelinux.0

• In the range options, there are two entries for the start file, which is linked to a policy in case a bios client is detected:
Configuration of a UCS Dhcp-Server

The following information comes from Univention itself:

9.6.13. opsipxeconfd configuration

Since opsipxeconfd 4.0.7.7 it is possible to configure the path of the files used as templates for UEFI netboot in the configuration file opsipxeconfd.conf.

This is possible through the options `uefi netboot config template x86` and `uefi netboot config template x64`.


Whether an UEFI BIOS meets the requirements of a client management system like opsi depends on several criteria. These criteria do not estimate the quality of the device, but only whether it can be managed by using netboot. This requires BIOS functions for UEFI netboot. Here an example comparison:

Table 17. Example for UEFI BIOS differences

<table>
<thead>
<tr>
<th></th>
<th>Lenovo Twist</th>
<th>MS-Surface</th>
<th>Dell Venue 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>UEFI pure</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>UEFI + CSM</td>
<td>√</td>
<td>x</td>
<td>√</td>
</tr>
<tr>
<td>Legacy</td>
<td>√</td>
<td>x</td>
<td>√</td>
</tr>
<tr>
<td>Both</td>
<td>√</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>UEFI Netboot</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>activatable entry</td>
<td>√</td>
<td>x</td>
<td>√</td>
</tr>
<tr>
<td>netboot without interaction</td>
<td>√</td>
<td>x</td>
<td>√</td>
</tr>
</tbody>
</table>

In this case `activatable entry` means, that for the next reboot a netboot can be activated by standard software. `netboot without interaction` means, that an activated netboot will be executed at the next reboot without any required interaction (like pressing any key combinations, F12 key, ...). If these preconditions are met, special opsi products can trigger a netboot. This feature is very important for automated processing. A product using this feature is for instance the localboot product for Windows and Linux `opsi-uefi-netboot`.

9.6.15. Technical details

The following sub chapters provide some information for scripted or manual handling of UEFI / GPT. For understanding how opsi works with UEFI/GPT, knowing these details is not required.

Technical details about UEFI

UEFI Bootloader entries can be managed on Linux with the program `efibootmgr`.

List of boot entries:

```
efibootmgr -v
BootCurrent: 000D
Timeout: 0 seconds
BootOrder: 0012,0011,000D,0010,000B,0009,0007,0008,000A,000C
Boot0000  Setup
Boot0001  Boot Menu
( .. )
Boot0007* USB CD
030a2400d23878bc820f604d8316c068ee79d25b86701296aa5a7848b66cd49dd3ba6a55
Boot0008* USB FDD
030a2400d23878bc820f604d8316c068ee79d25b6ff015a28830b543a8b864100961e49
Boot0009* ATA HDD0
030a2500d23878bc820f604d8316c068ee79d25b91af625956449f41a7b91f4f892ab0f600
Boot0009* PCI LAN
030a2400d23878bc820f604d8316c068ee79d25b7a84aaf2b2af4e79cf5cc8f3d3803
Boot0010* ubuntu    HD(1,800,31801,faaf7b9-bdf9-4767-b475-
0b8ae68d3ac)File(\EFI\ubuntu\grubx64.efi)
Boot0011* opsitempwinpe HD(4,3c72800,7cf801,dcc1ea68-a296-4fb8-a97a-
26327ed86f4)File(\EFI\boot\bootx64.efi)
Boot0012* Windows Boot Manager  HD(1,800,31801,5e4ffde2-3e25-42dd-b0f7-
fc7ee5d2b20)File(\EFI\Microsoft\Boot\bootmgfw.efi)WINDOWS.........x...B.C.D.O.B.J.E.C.T
.=.{.9.d.e.a.8.6.c.-..5.c.d.d.-..4.e.7.0.-..a.c.c.1.-
.f.3.2.b.3.4.4.d.4.7.9.5.}..a.............
```

On Windows UEFI boot loader entries can be managed with the program `bcdedit`.

List of boot entries:
Both programs can be used to create or delete entries, setting *netxboot* or change the boot order.
Example: Setting the entry for the next boot:

- Linux:
  
  ```
  efibootmgr /bootnext <hexId>
  ```

- Windows:
  
  ```
  bcdedit /set {fwbootmgr} bootsequence <GUID>
  ```

**Technical details about GPT**

GPT partitions know some new partition types. These are derived from the standard types. So the partition type for NTFS 07 becomes GPT 0700. The Linux partition types 82 and 83 become 8200 and 8300.

The list of known partition types can be shown:

```
# sgdisk -L
0700 Microsoft basic data      0c01 Microsoft reserved       2700 Windows RE
4100 PowerPC PReP boot         4200 Windows LDM data         4201 Windows LDM metadata
7501 IBM GPFS                   7f00 ChromeOS kernel         7f01 ChromeOS root
7f02 ChromeOS reserved         8200 Linux swap              8300 Linux filesystem
8301 Linux reserved            8302 Linux /home             8400 Intel Rapid Start
8e00 Linux LVM                 a500 FreeBSD disklabel      a501 FreeBSD boot
a502 FreeBSD swap              a503 FreeBSD UFS            a504 FreeBSD ZFS
a505 FreeBSD Vinum/RAID        a580 Midnight BSD data      a581 Midnight BSD boot
a582 Midnight BSD swap         a583 Midnight BSD UFS       a584 Midnight BSD ZFS
a585 Midnight BSD Vinum        a800 Apple UFS              a901 NetBSD swap
a902 NetBSD FFS                a903 NetBSD LFS             a904 NetBSD concatenated
a905 NetBSD encrypted          a906 NetBSD RAID           a900 Apple boot
af00 Apple HFS/HFS+            af01 Apple RAID             af02 Apple RAID offline
af03 Apple label               af04 AppleTV recovery       af05 Apple Core Storage
be00 Solaris boot              bf00 Solaris root           bf01 Solaris /usr & Mac Z
bf02 Solaris swap              bf03 Solaris backup         bf04 Solaris /var
bf05 Solaris /home             bf06 Solaris alternate se  bf07 Solaris Reserved 1
bf08 Solaris Reserved 2        bf09 Solaris Reserved 3     bf0a Solaris Reserved 4
bf0b Solaris Reserved 5        c001 HP-UX data             c002 HP-UX service
ea00 Freedesktop $BOOT         eb00 Haiku BFS               ed00 Sony system partitio
ef00 EFI System                ef01 MBR partition scheme    ef02 BIOS boot partition
fb00 VMWare VMFS               fb01 VMWare reserved       fc00 VMWare kcore crash p
fd00 Linux RAID
```

Actually the partition types shown in this list are just short forms for the actual GUIDs that are used. The partition schema is named after that.
So: 0700 stands for Microsoft basic data and for the GUID EBD0A0A2-B9E5-4433-87C0-686B72699C7

A list of GUIDs can be found at Wikipedia:

https://de.wikipedia.org/wiki/GUID_Partition_Table#Partitionstyp-GUIDs

https://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_type_GUIDs

Furtheron the tool gdisk (and sgdisk, ...) has an internal substitution table for unknown partition types. For the old partition type for vfat32 0b there is no corresponding 0b00. By passing the type 0b00 to sgdisk, it will be translated to 0700 without any message. Perhaps because of the consideration: vfat32 - this must be some Microsoft data partition ...

GPT partitionen can have attributes.

List of the currently known attributes:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Attribute value (sgdisk --info / diskpart gpt attribute)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nix</td>
<td>nix</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>0</td>
<td>system partition</td>
<td>0000000000000001</td>
</tr>
<tr>
<td>1</td>
<td>partition hidden from EFI</td>
<td>0000000000000002</td>
</tr>
<tr>
<td>2</td>
<td>legacy boot flag (legacy BIOS bootable)</td>
<td>0000000000000004</td>
</tr>
<tr>
<td>60</td>
<td>read-only</td>
<td>1000000000000000</td>
</tr>
<tr>
<td>62</td>
<td>hidden</td>
<td>4000000000000000</td>
</tr>
<tr>
<td>63</td>
<td>do not automount</td>
<td>8000000000000000</td>
</tr>
</tbody>
</table>

On Linux the attributes can be set with sgdisk by the option -A, --attributes and using the short form. On Windows they can be set with diskpart by the command gpt attributes and using the long form.

Examples:

```
select disk 0
select partition 1
gpt attributes=0x0000000000000000

```

show the partition table with -p, --print:
show detailed info for a partition (1) with `--info=`:

```
sgdisk --info=1 /dev/sda
```

**opsi UEFI/GPT Roadmap**

- UEFI 32 Bit support
- other netboot capable UEFI boot loader (grub2)

### 9.7. opsi with secureboot

OEM manufacturers deliver an additional layer of security with secureboot (sometimes written as *secure boot*). With secureboot only authorized Software and operating systems can be installed on a machine. This authorization is implemented directly via a key query to the UEFI. As long as the used keys match the operating system and software the usage will be authorized. The key inside the UEFI capable machines was created by Microsoft and the OEM manufacturer. Therefore any loaded efi binary or kernel has to be signed off by one of those keys. A *chain of trust* verifies that every loaded efi binary or kernel has a correct signature.

This module currently is a [co-funded opsi extension](mailto:info@uib.de).

Some preconditions are required to work with that module, should be duly noted that a suitable modules file is needed to unlock the feature, and also the prerequired module. You can get this unlocked modules file by purchasing the extension. For evaluation purposes we can provide you with an evaluation modules file free of charge. (→ mail to info@uib.de).

#### 9.7.1. Requirements

The technical requirements are as follows (with package versions):

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-linux-bootimage</td>
<td>&gt;= 20190923-4</td>
</tr>
<tr>
<td>opsipxeconfd</td>
<td>&gt;= 4.1.1.15-1</td>
</tr>
</tbody>
</table>

#### 9.7.2. Further remarks for the usage of the secureboot module

Secure boot is only possible on clients that support this feature as part of their firmware. The installation of a client with secureboot is only supported on 64-bits.

UEFI Netboot installs require a winpe that is capable of booting in UEFI mode. Often, an existing
winpe will be capable of booting in such way, to verify it check if there is a folder named EFI, as well as a file named bootmgr.efi inside your winpe folder. If that is not the case, create a UEFI compatible winpe as explained in our opsi-getting-started Manual, Chapter "Creating a PE". The winpe, that is UEFI capable, needs to be located in the winpe_uefi folder of the opsi netboot product. If your winpe is already capable of booting UEFI and MBR modes, you could simply place a softlink winpe_uefi > winpe.

9.7.3. opsi server configuration for secureboot clients

If you have an external DHCP server you have to configure it so it forwards an UEFI Netboot to your opsi-server. The bootfile has to be: linux/pxelinux.cfg/shimx64.efi.signed

Activate the “Uefi-Boot” checkbox in the opsi-configed for uefi clients. Alternatively you can run the following command:

```bash
opsi-admin method configState_create "clientconfig.dhcpd.filename" "<insert host id here>" "linux/pxelinux.cfg/elilo.efi"
```

Using the file elilo.efi via the opsi-admin command only serves the purpose to set the UEFI checkbox within the opsi-configed. This has no relation to the usage of the dhcp bootfile. The opsi-configed will feature a secureboot checkbox and accept the proper filename in a future release.

In addition the UEFI template for UEFI clients has to be changed in the configuration file of the opsipxeconfd. The specific uefi netboot config template x64 must have the value /tftpboot/linux/pxelinux.cfg/install-grub-x64. We recommend to run the opsi-setup --init-current -config command after this change.

All UEFI clients will boot the Microsoft signed file shimx64.efi.signed which is provided by the opsi-linux-bootimage package. Clients with activated secureboot verify the signature and load the grub2 bootloader file. Non secureboot clients fail this verification and also load the grub2 bootloader. Either way both type of clients load the shimx64.efi.signed file. The installation itself doesn't differ in secureboot and non secureboot mode. Clients successfully installed in secureboot mode can be checked with the command msinfo32. The output contains an entry secureboot.

9.7.4. Client configuration

Please keep in mind that terms might change depending on manufacturer and hardware model,

The UEFI of the client should be configured as following:

- Secureboot enabled
  This entry is most commonly found in the section Boot, Startup of Security.

- Run the BIOS in UEFI mode.
If you have a choice between **UEFI only**, **Legacy only** and **Both** choose **UEFI only**. Deactivate **Legacy Support** as Secure Boot only works on a **UEFI only** setup. **CSM Support** in combination with **UEFI only** can be left activated. In case of doubt deactivate the **CSM Support** feature. **UEFI Network Boot** needs to be activated. This entry can also be called **UEFI Network Stack**. If your machines UEFI gives you a distinction between **IPv4** and **IPv6**, **IPv4** is the choice to make.

### 9.8. opsi local image

#### 9.8.1. Requirements for the opsi extension opsi local image

This module is currently a **cofunding project**.

Some conditions have to be met to use this module. This means that you need an activation file to use it. You will receive this activation if you buy the extension. For evaluation purposes, we also provide a time-limited activation free of charge (→ mail to **info@uib.de**).

Further details can be found in Section 9.1, “Activation of non-free modules”.

With the permission to use opsi-local-image you also acquire the right to use the extension opsi-vhd-reset (see Section 9.9, “opsi vhd reset”).

A technical requirement is opsi 4.0.3 with package versions:

<table>
<thead>
<tr>
<th>Table 19. Required packages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>opsi package</strong></td>
</tr>
<tr>
<td>opsi-linux-bootimage</td>
</tr>
</tbody>
</table>

For the product **opsi-local-image-capture** the share **opsi_depot_rw** must have write permission for **pcpatch**. Check your Samba configuration.

#### 9.8.2. Introduction

Opsi offers a good basis for the automated installation and maintenance of Windows computers - especially when it comes to heterogeneous hardware. The package-based installation technology supported by opsi is not fast enough to use computers in training rooms within a short time, e.g. to put it back into a defined state during a break between two courses. Therefore, a concept is presented here in which the package-based installation is saved locally on a second partition as an image copy and from there a quick recovery is possible.

1. Initial installation finishing with a local image backup
2. Fast recovery based on different techniques
3. System maintenance finishing with a local image backup
4. Integration of capturing of existing installations to WIM
5. Integration of Linux clients in the Backup/Restore process.
9.8.3. Concept

The requirements of educational computer networks differ in part from those of other networks. An essential requirement, which is discussed in the following part, is the quick restoration of computers to a defined state, which has been changed because of a temporary use. Specifically, it is about the provision of computers in class rooms, whereby the problem also applies to commercial computer rooms or university computer pools.

The restore should take place within a short time (approx. 15 minutes) and should, as far as possible, not only reset a computer but also switch to another base installation (e.g. Win XP / Win 7 / Linux). It must also be possible to ensure that the systems are continuously updated with security updates.

The usual techniques for installing PCs have several specific advantages and disadvantages:

*Table 20. Advantages and disadvantages of unattended and image based solutions*

<table>
<thead>
<tr>
<th>Feature</th>
<th>Unattend</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>(-) slow</td>
<td>(+) fast</td>
</tr>
<tr>
<td>Sensitivity to heterogeneous Hardware</td>
<td>(+) low</td>
<td>(-) high</td>
</tr>
<tr>
<td>Network load</td>
<td>(-) high</td>
<td>(-) high</td>
</tr>
</tbody>
</table>

The concept of opsi-local-image tries to combine the advantages of both approaches:

*Table 21. opsi-local-image*

<table>
<thead>
<tr>
<th>Feature</th>
<th>Unattend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>(+) fast</td>
</tr>
<tr>
<td>Sensitivity to heterogeneous Hardware</td>
<td>(+) low</td>
</tr>
<tr>
<td>Network load</td>
<td>(+) low</td>
</tr>
</tbody>
</table>

The concept can be summarized in the following key points:

- Initial Windows installation via PXE boot, package-based with individual driver integration using the opsi-Linux-Bootimage
- Store this initial installation in a backup image on another partition on the local hard drive using the opsi-Linux-Bootimage
- Quick restore the productive installation from the local image using the opsi-Linux-Bootimage
- Maintenance of the local installation (security updates) via the opsi software distribution and backup of the updated system to the local backup image using the opsi-Linux-Bootimage

9.8.4. Technical Concept

The workstation is used with a static partition table. You can work with either 3 or 4 partitions:
• **Partition 1 (System)**
  Holds the currently used operating system (Windows / Linux)
  The size of this partition is set during partitioning via a property of the product `opsi-local-image-prepare`.

• **Optional: Partition 2 (sysdata)**
  This can be for example be used to store user data, which is not overwritten during a restore. The formatting is NTFS.
  The size of this partition is set during partitioning via a property of the product `opsi-local-image-prepare`.

• **Partition 3 (winpe / swap)**
  The size of this partition is static and set to 4GB.
  This partition is not used on Windows XP.
  On NT6 (Windows 7) this partition is used for the winpe required during installation and is not visible during normal operation.
  On Linux this partition is used as swap.

• **Partition 4 (backup)**
  This partition is used to store the backup images and their metadata.
  The size of this partition is the free space still available after the other partitions have been created.

The netboot products for operating system installations use only the first two or three partitions (XP uses only the first) and leave the last backup partition untouched. This means that the images on partition 4 (backup) are retained even when installing a new operating system.

### 9.8.5. Process steps

**Initial Installation**

The necessary static partitioning is first created using the product `opsi-local-image-prepare`.

![Figure 87. schema: static partitioning with opsi-local-image-prepare](image)

Then the products `opsi-local-image-win*` or others can be used to install several operating systems with different configurations and different application software.
By default, these are automatically saved as an image after installation.

Restoring an image

Executing the product `opsi-local-image-restore` automatically restores the last created image. If a different image is to be restored, this has to be specified in the property `imagefile`.

Deleting an image
By executing the product `opsi-local-image-delete`, the image specified in the property `imagefile` will be deleted.

**Updating an image: automatic work flow with opsi-auto-update**

`opsi-auto-update` is a product to simplify the maintenance of the clients.

In essence, this product can be used to ensure that the installed products are up to date. The product sets all installed products, whose version is not identical to that on the server, for the client to setup.

Since this product can not only be used in the context of `opsi-local-image`, it is described in the chapter `opsi standard products / opsi-auto-update`: Section 8.1.1.19, “opsi-auto-update”

### 9.8.6. The opsi-local-image products

The opsi-local-image products from version 4.1 also support systems with more than one harddisk. Please also note the section Section 8.2.3, “Some hints to the NT6 netboot products (Win7 to Win 10)”

The `opsi-local-image` package consists of the following products

The netboot product for partitioning

- `opsi-local-image-prepare`

The netboot products for OS installation:

- `opsi-local-image-winxp`
The netboot products for handling the local images

- opsi-local-image-backup
- opsi-local-image-restore
- opsi-local-image-delete

The localboot products for process control:

- opsi-local-image-backup-starter
- opsi-auto-update

To install these products please set the attribute **active** of the repository **uib_local_image** in the file /etc/opsi/package-updater.repos.d/uib-local_image.repo to **True**. Executing `opsi-package-updater --repo uib_local_image repo install` will then install the new products.

**UEFI Compatibility**

The opsi-local-image products are compatible with UEFI.

**netboot product for partitioning**

- opsi-local-image-prepare
  
  Creates the static partitioning of the harddisk for all other products.

  **Properties:**

  **ask_before_inst**
  
  Determines if the start of the installation has to be confirmed on the client. (Default=**true**)

  **system_partition_size**
  
  Determines the size of partition 1 (system). (Default = 30G)

  **data_partition_size**
  
  Determines the size of partition 2 (data). If set to **0G**, no partition will be created for data. (Default = **0G**)
start_os_installation

The product to install an operating system can be selected here, which is started automatically after the partitioning. When installing any of these products, the product properties `imagefile` and `imagefiles_list` are deleted for the product `opsi-local-image-restore`, as this data has become invalid due to the repartitioning.

delay_for_reboot

Seconds between the end of the script and the reboot to give the server time to create the netboot pipes.

minimal_backup_partition_size

This property is used to check if the size entries make sense. The size of the backup partition results from:

\[
\text{Harddisk size} - (\text{system_partition_size} + \text{data_partition_size} + \text{winpe_partition_size}).
\]

Usually opsi-local-image is used because a local backup of the system partition is to be made. This requires that there is enough space for the backup partition. If, during the calculation of the partitioning, it is determined that the remaining space for the backup partition is smaller than the value of `minimal_backup_partition_size`, the process is aborted. (Default=55%)

winpe_partition_size

Size of the winpe partition (Default=4G)

multi_disk_mode

This property is used to select the target disk of the installation.

Possible values are: "0", "1", "2", "3", "prefer_ssd", "prefer_rotational"

The values "0", "1", "2", "3" indicate the direct index of the harddisk ("0"= first harddisk)

The value "prefer_ssd" selects the first SSD.

The value "prefer_rotational" selects the first classic (rotational) disk.

This property is ignored on systems with only one disk.

Default = "0"

backup_partition_on_same_disk

true : create the backup partition on the system disk. false : create the backup partition on the first disk that is not the system disk.

This property is ignored on systems with only one disk.

Default = "true"

⚠️ Use this product only for the initial preparation of the disk. It deletes all existing images.

netboot products for the installation of Windows

The special netboot products for installing Windows are derived from the opsi standard products for installing Windows. Specifically, this means that they are identical in terms of structure and driver
integration. Corresponding instructions can be found in the *opsi-getting-started* manual.

The properties of the Windows NT6 products from version 4.1 are a subset of the properties of the standard NT6 products as described in the Section 8.2.3, “Some hints to the NT6 netboot products (Win7 to Win 10)” section. The missing properties for disks and partitions are taken from the product opsi-local-image-prepare.

**ATTENTION**

Do not change the property values of opsi-local-image-prepare after you have prepared the machine with it, as subsequent products will access these values.

- **opsi-local-image-winxp**
  Installation of Windows XP. Uses the first partition only. Administrator password is empty.

- **opsi-local-image-win7**
  Installation of Windows 7 32 Bit.

- **opsi-local-image-win7-x64**
  Installation of Windows 7 64 Bit.

- **opsi-local-image-win81**
  Installation von Windows 8.1 32 Bit.

- **opsi-local-image-win81-x64**
  Installation von Windows 8.1 64 Bit.

- **opsi-local-image-win10**
  Installation von Windows 10 32 Bit.

- **opsi-local-image-win10-x64**
  Installation von Windows 10 64 Bit.

These products have the following specific properties for *opsi-local-image*:

- **backup_after_install** with the default value *true*. In this case, this means that after the OS installation, the application software is first installed and then an image backup of the installation is created. Furthermore, the value of the *imagefile* of the product *opsi-local-image-restore* will be deleted. This means that the created backup will be named like the running netboot product (e.g. *opsi-local-image-win7*).

- **setup_after_install**
  One or more products can be specified here, which are set to *setup* after the operating system installation is complete. The dependencies of these products are also resolved.

**Netboot products for installing Linux**

- **opsi-local-image-ubuntu**
  Installation of Ubuntu Linux 12.04/14.04 32Bit/64Bit.
  The installed system has two users: *root* and *user*. The password for *root* will be set according to the product property *root_password* (default: *linux123*). For *user* the password will be set according to *user_password* (default: *linux123*). Details of the installation can be configured with product
properties. The main product properties are:

- **askbeforeinst:**
  Determines if the start of the installation has to be confirmed on the client. (Default=\textit{true})

- **architecture:**
  Architecture selection, determines the selection of the bootimage and the installation architecture. (Default=\textit{64bit})

- **additional_packages:**
  Which additional packages should be installed? Specification of packages separated by spaces. (Default = pass: [""])

- **language:**
  Determines which language / locale should be installed. (Default=\textit{de})

- **console_keymap**
  Determines which keyboard layout should be installed. (Default=\textit{de-latin1-nodeadkeys})

- **timezone:**
  Determines which timezone should be used. (Default=\textit{Europe/Berlin})

- **online_repository**
  Specifies from which online repository the packages should be installed. The default is \texttt{http://de.archive.ubuntu.com/ubuntu}

- **proxy:**
  Proxystring (if required) with the following syntax: \texttt{http://<ip>:<port>} (Default = ")

- **backup_after_install**
  (\texttt{true}/\texttt{false}) Default = \textit{true}. After the installation, an image backup of the installation is created.

- **setup_after_install**
  One or more products can be specified here, which are set to \textit{setup} after the operating system installation is complete. The dependencies of these products are also resolved.

- **wget_and_execute:**
  Url (http) of a file which is fetched and executed at the end of the installation. (Default = pass: ["])

- **release:**
  Determines which release of Ubuntu should be installed. (Default = "\texttt{trusty}")

- **install_opsi-client-agent:**
  Specifies if the Linux opsi-client-agent should be installed. (this is a cofunding project and requires activation via /\texttt{etc/opsi/modules}). (Default=\textit{false})

### Netboot products for backup and restore

- **opsi-local-image-backup**
  This product backs up the operating system currently installed on partition 1 in an image file on partition 4. The name set in the product property is used as the name for the image. If this is empty, the name of the opsi netboot product is used, which is currently set to \textit{installed} (e.g. \texttt{opsi-local-image-winxp}). This name is set for the product \texttt{opsi-local-image-restore} as the product
property `imagefile`, so that a subsequent call to `opsi-local-image-restore` will restore exactly this image by default. This name is added to the product property `imagefiles_list` for the product `opsi-local-image-restore`. This property contains the list of all available images. Furthermore (for the Windows products) the current opsi product versions are saved together with the image so that they can be restored as part of the restore.

The backup tool which is used is partclone.

Properties:

- **askbeforeinst:**
  Determines if the start of the installation has to be confirmed on the client. (Default = `false`)

- **free_on_backup:**
  This is a read-only property that shows current information about the backup partition: device, size, used, remaining, use in percentage, mount point.

- **imagefile**
  Name of the image file to be created (default = empty = the name of the currently installed opsi-local-image product will be used). The name may contain spaces but no umlauts/tremas. (In the case of spaces, these are treated internally as underscores, i.e. `my image = my_image`).

- **setup_after_install**
  One or more products can be specified here, which are set to `setup` after the operating system installation is complete. The dependencies of these products are also resolved.

- **opsi-local-image-restore**
  This product restores the image specified in the `imagefile` product property to partition 1 and marks it bootable. Furthermore (for the Windows products) the opsi product versions saved with the image are restored together with the image.

Properties:

- **askbeforeinst:**
  Determines if the start of the installation has to be confirmed on the client. (Default= `true`)

- **architecture:**
  Architecture selection, determines the selection of the bootimage and the installation architecture. (Default= `64bit`)

- **imagefile**
  Name of the image to be restored. The value of this property has automatically been set by the last backup. The list of available images can be found in the property `imagefiles_list`.

- **imagefiles_list**
  List of the available images. This list is maintained by the backup product.

- **update_and_backup**
  **The use of this property is discouraged**
  Instead, use the product `opsi-auto-update`. This product is described in the chapter `opsi standard products / opsi-auto-update`:
  Section 8.1.1.19, “opsi-auto-update”

  (true/false) Default = `false`. If set to `true`, after the restore, it is ensured that all localboot products that are available on the server in a different version are set to `setup` and the product `opsi-local-image-backup-starter` is set to `once`. This means that all existing updates
are installed and a backup is created automatically after the updates.

- **setup_after_restore**
  One or more opsi products can be specified here, which are set to *setup* after the restore is completed and are therefore automatically installed after the reboot. The default is the product *windomain* to add the restored client to the Windows domain again.

- **opsi-local-image-delete**
  This product deletes the image given by the product property *imagefile* from the backup partition

  - **imagefile**
    Name of the image to be deleted (default = empty, results in an error when executing)

**Localboot products for process control**

- **opsi-local-image-backup-starter**
  This localboot product sets the Netboot product *opsi-local-image-backup* to *setup* and reboots the client. This product has a very low priority of -98. This means, that all usual localboot products will be installed first.

- **opsi-auto-update**
  In essence, this product can be used to ensure that the installed products are up to date. The product sets all installed products, whose version is not identical to that on the server, for the client to setup.

Since this product can not only be used in the context of *opsi-local-image*, it is described in the chapter *opsi standard products / opsi-auto-update*:
Section 8.1.19, “opsi-auto-update”

### 9.8.7. Extended opsi service methods

With this extension, the computers in a training room can be combined in an opsi-client group. In order to provide the most convenient management for all computers in a training room, the following extensions to the opsi-service methods have been implemented:

- **setProductActionRequestForHostGroup**
  Parameter: *hostGroupId, productId, actionRequest*
  Makes it possible to start a specific action (e.g. restore an image) for all members of a group (e.g. computers in a training room).

- **setProductPropertyForHostGroup**
  Parameter: *productId propertyId propertyValue hostGroupId*
  Makes it possible to set a given product property (e.g. which image is to be restored) for all members of a group (e.g. computers in a training room).

- **getPossibleImagefileValuesForHostGroup**
  Parameter: *groupId*
  Returns the list of image filenames which *opsi-local-image-backup* has created on all members of the group. If a certain image (e.g. *opsi-local-image-winxp*) is not available on one or more computers, it is not an element of the returned list.
These methods will be integrated into the standard opsi packages at a later date. Until then, a file 40_groupActions.conf is available, which must be copied with root rights to /etc/opsi/backendManager/extend.d. Then execute the following: opsi-setup --set-rights /etc/opsi.

9.8.8. Backup partition

The backup partition is (with MBR BIOS and without data partition) the third partition of the system hard disk.

On systems with more than one disk, the system hard disk is determined by the opsi-local-image-prepare property multi_disk_mode.

On systems with more than one disk, the backup partition can also be the first partition of another disk, depending on the opsi-local-image-prepare property backup_partition_on_same_disk.

It contains:

- The file master.log with information about all performed image operations. This log file is transferred to the bootimage logs.
- The image directories
  The image directories have the same name as the image and contain the image as well as the metadata of the image.
  To give an idea about file sizes, here are the sizes of different image directories with OS and standard software (libreoffice, adobereader, firefox, thunderbird, javavm, flashplayer):
  - opsi-local-image-ubuntu: 3.6G
  - opsi-local-image-winxp: 6.4G
  - opsi-local-image-win7: 9.4G
  - opsi-local-image-win7-x64: 13G

9.8.9. Capture Images (WIM) generating and distribution

Capture Images (WIM) Introduction

Starting with NT6 (i.e. from Vista) Microsoft has introduced a new image format, the Windows Imaging Format (WIM). A WIM image is no longer a disk or partition image, but rather a file and metadata archive. A WIM file can contain multiple images. The standard installation of a NT6 computer is based on the setup.exe extracting an image from the install.wim file and then configuring it and providing it with additional drivers.

The installed Windows OS including installed software, hotfixes and configurations can be read from an existing computer and saved in the form of a WIM. Such a WIM can then be the basis for new installations.

Capture Images (WIM) Components

To create a capture image in Wim format, from version 4.1 you only need the product:

- opsi-local-image-wim-capture
The previous products:

- opsi-local-image-capture
- opsi-local-image-sysprep

are obsolete and can be removed.

In addition, there are the target products which are intended to hold the captured images:

- opsi-local-image-win7-capture
- opsi-local-image-win7-x64-capture
- opsi-local-image-win81-capture
- opsi-local-image-win81-x64-capture
- opsi-local-image-win10-capture
- opsi-local-image-win10-x64-capture

Capture Images (WIM) Processing

The process and settings for the product opsi-local-image-wim-capture are very similar to the product opsi-wim-capture which is described here: Section 9.4, “opsi WIM Capture”. The properties of opsi-wim-capture are described here: Section 9.4.6.1, “Main Product opsi-wim-capture”.

The main difference between the two products is:

To backup and restore the partition to be captured opsi-local-image-wim-capture uses the mechanism of opsi-local-image-backup/opsi-local-image-restore. For the same purpose opsi-wim-capture uses the product opsi-clonezilla.

opsi-local-image-wim-capture will fail if you have setup your system with a data partition. In this case, reinstall the computer with the opsi-local-image-prepare property data_partition_size=0.

9.8.10. Windows installation from a target product

(Roll out the captured image)

Restore opsi metadata about installed products

The Problem:

If you reinstall Windows with opsi, e.g. win7-x64, then during the installation of the opsi-client-agent all local boot products that were previously set to installed on this computer are automatically set to setup and thus reinstalled later.

This cannot be done in the same way when rolling out a captured image. The image contains the backup of the opsi data that was stored there during the capture process. This is discovered during the installation of the opsi-client-agent and imported back into the opsi-server.
The products that were installed in the captured image are now set to installed for the freshly installed computer. If all products set to installed were now set to setup, this would result in all products already installed in the image being installed again. This is not desirable.

As of opsi 4.0.7, there are two options for restoring the opsi metadata for installed products:

**Option 1**

Restore metadata and retain setup action requests.
Products that have the status installed will not be set to setup.
This is the default and the behavior before opsi 4.0.7

**Option 2**

Restore metadata. Products that are marked as installed are set to setup except those which were contained in the restored metadata.

**Option 1**

When rolling out a captured image, only those products are automatically installed that were set to setup before the start of the operating system installation. This can be changed manually or with the property setup_after_install. In this case, therefore, only those products are installed that were set to setup before the operating system was installed.
This is the default and the behavior before opsi 4.0.7

**Option 2**

Option 2 behaves similarly to installations from non-captured images:
* Restore the metadata.
* Products that are marked installed are set to setup except those which were contained in the restored metadata.

This behavior is only available from opsi 4.0.7 and is not the default. Option 2 has been made possible by extensions to the opsi-script and is part of the opsi-client-agent from 4.0.7.
To use this behavior a config (host parameter) must be set:
The Boolean configuration entry: clientconfig.capture.switch_installed_products_to_setup. If this entry has the value true for the client then option 2 is used, otherwise option 1.

These host parameter can then be used to activate or deactivate client-specific events. The host parameter can be created using opsi-configed or opsi-admin.

To create the host parameter using opsi-admin, the following commands must be executed on the opsi-config-server:

```bash
opsi-admin -d method config_createBool
clientconfig.capture.switch_installed_products_to_setup
"capture.switch_installed_products_to_setup" true
```

This sets Option 2 for all computers.

To create the host parameter using opsi-configed, select Server configuration / clientconfig / Right-click on the right side: Add Boolean configuration entry.
9.8.11. Helper product opsi-wim-info

The product opsi-wim-info can be used to quickly read information about the images saved in an install.wim. This information is then saved in the log file.

Properties:

- **target_product**
  ProductId of the product where is searched for install.wim.

9.8.12. Creating your own Ubuntu proxy

You can find useful instructions for creating your own Ubuntu proxy here:

- [http://wiki.ubuntuusers.de/Lokale_Paketquellen/Apt-Cacher-ng](http://wiki.ubuntuusers.de/Lokale_Paketquellen/Apt-Cacher-ng)

9.9. opsi vhd reset

9.9.1. Requirements for the opsi extension opsi vhd reset

This module is currently a cofunding project.

It is bundled with the extension opsi-local-image (see Section 9.8, “opsi local image”) - that means: the activation for opsi-local-image also applies to opsi-vhd-reset.

Some conditions have to be met to use this module. This means that you need an activation file to use it. You will receive this activation if you buy the extension. For evaluation purposes, we also provide a time-limited activation free of charge (→ mail to info@uib.de).

Further details can be found in Section 9.1, “Activation of non-free modules”.

As a technical requirement opsi >= 4.0.7 is required with the package versions:

<table>
<thead>
<tr>
<th>opsi package</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-winst</td>
<td>&gt;= 4.12.0.13</td>
</tr>
</tbody>
</table>

9.9.2. Introduction

To use computers in training rooms within a short time, e.g. to put them back into a defined state during a break between two courses requires special techniques. With opsi-local-image opsi already offers a solution, that is now supplemented by a new method that has specific advantages and disadvantages.

1. Initial Windows 10 installation in a VHD container.
2. Sealing the initial installation by using a child VHD.

3. Quick restore by replacing the child VHD.

4. Upgrade of the initial installation by a merge of the child VHD.

5. This method works with snapshot techniques known from virtualization without using virtualization itself.

9.9.3. Process steps

Initial Installation

Windows 10 is installed in a VHD container using the product `opsi-vhd-win10-x64`.

Figure 93. Scheme: Initial Installation 1: Creation of the VHD

Figure 94. Scheme: Initial Installation 2: Windows Installation

The desired applications can then be installed on this Windows.
By executing the opsi product `opsi-vhd-control`, the current opsi metadata for this client (which product is installed in which version) is stored in the initial installation. The Windows PE is then activated and booted for the further process. The product `opsi-vhd-control` has a very low priority (-97) and therefore only runs after the installation of application software. As a result, the `opsi-vhd-control` product can be set to `setup` together with the application software.

After the Windows-PE boots, the second part of `opsi-vhd-control` starts to work and creates a child VHD which seals the initial installation and records all further changes.
All changes from now are stored in the *child* VHD.

![Diagram: Working with the sealed system](image)

**Figure 98. Scheme: Working with the sealed system**

**Fast recovery**

The initial installation can be restored using the opsi product *opsi-vhd-control*. First, the saved opsi metadata is restored from the system. Then for handling of the Child VHD the system boots into Windows PE again.

![Diagram: opsi-vhd-control: Recovery of the initial installation 1](image)

**Figure 99. Scheme: opsi-vhd-control: Recovery of the initial installation 1**

The *child* VHD with the changes is deleted in Windows PE and replaced with a new, empty *child* VHD.

![Diagram: opsi-vhd-control: Recovery of the initial installation 2](image)

**Figure 100. Scheme: opsi-vhd-control: Recovery of the initial installation 2**
Updating an image using **opsi-auto-update**

To update the initial installation with patches and software updates, you can proceed as follows:

- Restore the initial installation (as described above)
- Install the updates
- Integration of the updates in the initial installation and re-sealing by starting **opsi-vhd-control** with the property `upgrade=true`
- This also stores the new opsi metadata in the system

These processes are carried out automatically by the product **opsi-auto-update**.

The product **opsi-auto-update** replaces the previous product **opsi-vhd-auto-upgrade**

### 9.9.4. The opsi-vhd products

The extension **opsi-vhd-reset** consists of the following products:

The netboot product for the initial installation:
- **opsi-vhd-win10-x64**

The localboot product to control the creation, the replacement and merge of the child VHDs:
- **opsi-vhd-control**

The localboot product to control the fully automatic upgrade of the parent VHD.
- **opsi-auto-update**

**UEFI Compatibility**

The opsi-vhd products are UEFI compatible.

**The opsi netboot product** **opsi-vhd-win10-x64** **and its properties**

This netboot product is very similar to the normal netboot products (4.1.0) for Windows installations and must be filled accordingly as described in the **getting-started** manual. Also the properties are mostly the same.

The following properties are special for this product:

- **windows_vhd_size**
  This property specifies the size of the base VHD absolute or as a percentage of the harddisk size, minus the WinPE partition. The default value of 100% is automatically reduced to 80% to leave room for the child VHD. If (absolute or relative) a value is entered that would exceed 80%, this is also reduced to 80%.
  This property replaces the standard property **windows_partition_size**
(Default = 100%)

- **installto:**
  This value is vhd and should not be changed.

The following properties are not existing for this product:

- **windows_partition_size, windows_partition_label**
  See above. The label of the partition on which the VHDs are located is CONTAINER.

- **data_partition_size, data_partition_letter, data_partition_create, data_partition_preserve**
  Currently the usage of a data partition is not possible with opsi-vhd.

- **boot_partition_size, boot_partition_letter, boot_partition_label**
  Currently the usage of a boot partition is not possible with opsi-vhd.

- **pre_format_system_partitions, preserve_winpe_partition**
  With opsi-vhd these two values are always true.

**The opsi localboot product opsi-vhd-control and its properties**

The opsi-vhd-control product has a very low priority (-96).

- **disabled**
  This property is for debugging purposes.
  If true, the product does not execute any actions.
  Default = false

- **upgrade**
  If true: Merge the changes that are collected in the child VHD to the parent VHD. Then replace the child VHD with an empty one.
  If false: replace the child VHD with an empty child VHD.
  At the end of a successful upgrade run, this property is automatically reset to false.
  Default = false

- **stop_on_no_network_in_pe**
  This property is for debugging purposes.
  If true: Abort with an error message, to analyze why no network connection could be established.
  Default = false

**The opsi localboot product opsi-auto-update and its properties**

opsi-auto-update is a product to simplify the maintenance of the clients.

In essence, this product can be used to ensure that the installed products are up to date.
The product sets all installed products, whose version is not identical to that on the server, for the client to setup.
Since this product can not only be used in the context of opsi-vhd-reset, it is described in the chapter opsi standard products / opsi-auto-update:
Section 8.1.1.19, “opsi-auto-update”
Known Problems and Restrictions

- There is also a 32 bit version. Due to a problem with the diskpart merge command, this can only be used to a limited extent in the 32 bit Windows PE versions.
- In theory, an implementation for Windows 8.1 or Windows 7 Enterprise would be possible. We will only build these products at request.
- There are indications that a Windows 10 release upgrade of an installation in a VHD will fail. ([https://www.heise.de/newsticker/meldung/VHD-Boot-Windows-Update-demoliert-Aktivierung-3806023.html](https://www.heise.de/newsticker/meldung/VHD-Boot-Windows-Update-demoliert-Aktivierung-3806023.html))

9.10. opsi License Management

9.10.1. Conditions for using the opsi License Management extension

This module currently is a co-funded extension and therefore not free. For more details see Section 9.1, “Activation of non-free modules”.

9.10.2. Overview

Main features

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

The main features are:

- Providing license management functions from within the opsi-configed management interface.
- Easy generation of reports about installed license-bound software, for non-opsi-managed software, based on the software inventory
- Reconciliation of software installations and license contracts
- Optionally automated supplying, assigning, and reserving licenses and license keys.
- The following license models can be used:
  - standard single license - one installation, possibly identified by a license key, on one computer
  - volume (or campus) license - a certain or unlimited number of installations for which one license key is provided
  - computer bound license - a single license for use on a fixed hardware
  - concurrent license - the installations are not limited, the real usage is managed by a license server
- When uninstalling an opsi managed software the assignment of a license (key) is released as well.
- In addition, license assignments can be edited manually or even by a script, for example, if licenses for software that is not opsi-managed should be integrated.
Overview database model

In the world of non-free software, license management is a complex subject. To represent it, opsi has to implement a relatively complex database model.

The involved tables in the opsi database are sketched in the following diagram. The meaning of the different relations should *peu à peu* become clear by the following explanations of the opsi LICENSE management concepts and usages.

The blue line in the diagram marks the border of the tables which are automatically generated by the software audit functions and the data which is constructed especially for license management. Only the license pool table has connections to both spheres. This fact hints at the importance of this construct.

![Diagram of database tables relevant for license management](image)

**Figure 101. Database tables relevant for license management**

**Invoking the license management from the opsi-configed**

A separate window in the *opsi-configed* management GUI is used for the license management. It is available by pressing the button "licenses" at the top right corner of the *opsi-configed* management GUI. If the license management module is disabled, then a note will be displayed. (See the entry for "license management" in the main menu under /Help/Modules).
The opsi license management module is an opsi extension module.

9.10.3. license-pools

The license management frame is constructed similarly to the mainframe, showing a line of tabs that lead to subframes, each subframe handling a specific task.

The first subframe introduces the topic "license pools", and defines their relationship to the data on installed software and opsi packages.

What is a license-pool?

A license-pool has to be defined if licensing is needed to be managed for some kind of software.

The license-pool is a construction that denotes the collection of all allowances for installing some kind of software.

The license-pool is the central element of the opsi license management since it encloses two sides, on one hand, references to installed software items and opsi installation packages, on the other hand, references to the legal and technical constructions of the installation permissions.
Administration of license-pools

At the top of the license-pools window is a table of available license-pools.

The field description can be edited here. More editing functions are available from the context menu (right mouse button). The most important is: creating a New license-pool by adding a new row to the table.

When inserting a row, a (unique) licensePoolId must be entered, e.g. following the scheme softprod_pool.

Please do not use special characters. When saving the new entry, any capital letter will be converted to lower case, therefore avoiding capitals from the beginning is recommended.

The new licensePoolId cannot be changed after it is saved because it is used as the primary key.

After any changes to the data in the window, the green checkmark changes to red, and the cancel option is enabled. The changes can be saved by clicking the red checkmark, or changes can be canceled by clicking the cancel button (also available from the context menu).
license-pools and opsi-products

The standard method to manage licenses is to include the license, from a single license-pool, when installing the software (i.e. using the opsi-product installation software to install Acrobat Writer).

Not unusual is the case that multiple products share the same license pool. This is normal if these products are variants of the same software. For example: the products win10-x64 and opsi-local-image-win10-x64 using the license pool p_win10-x64).

A more complicated situation (which you should avoid) might occur while installing software that requires licenses from several license-pools (i.e. "Designer tools" which installs Adobe Photoshop as well as Acrobat Writer). In this case, the opsi-product requests licenses from several license-pools. At the same time, there might be other opsi-products requesting licenses from the same license-pools (e.g. the Acrobat Writer license-pool). So the relation between opsi-products and license-pools can be ambiguous. This can be avoided by using unambiguous policies when building opsi-products.

Do not integrate more than one license needing software in one opsi product. Assign this product to the license pool which holds the licenses for this product. (Without this assignment the license management will not work together with the opsi WAN extension. See also chapter Section 9.11, “opsi WAN/VPN extension”)

The second part of the license-pool tab manages the relationship between license-pools and productIds (from opsi-products).

All tables in the license management module can have their columns sorted by clicking on the column header. Clicking again inverts the order (ascending or descending).

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

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

license-pools and installed software

The third section of the license-pools tab deals with the correlation between license-pools and installed software found on computers in the opsi network, either installed via opsi or otherwise.

By default, the table in this section contains the identifying attributes of all captured software items. The ID, which is constructed from combining all attributes, acts as a unique key for every software item installed on any computer. The values are determined and transferred to the server via opsi software audit. They are saved in the SOFTWARE_CONFIG table of the opsi database and added to the collective SOFTWARE table, provided they are not yet present. This database table is the base for the table shown in the license management and is of importance outside of the license management as well.
The display of the software table is configured by two lists of options. The first one enables the possibility to restrict the display on certain data:

- Show all software names
- Show software names without uniform assignments
- Show software names without any assignments

The other defines what the selection of a row entails:

- Selection = Complete list of SW items assigned/to assign
- Selection = SW items to assign (additionally)

![Figure 104. Software table configuration](image)

The first option in both lists is always set by default. The software items (from the SOFTWARE database table) that are assigned to the selected license-pool (from the LICENSE_POOL table) are shown by highlighting the respective rows in the tables. This allocation is precisely the data from the AUDIT_SOFTWARE_TO_LICENSE_POOL table.

The assignment of software items to the license-pool can be edited by changing the highlighted rows:

The assignment of software items to license-pools is edited by changing the various selections, either with Ctrl + click or Shift + click. A single mouse click resets the selection.

If there is an entry for an allocation of a software item to a license-pool in the database but the software does not seem to exist anymore, then the Missing button is set to active. By clicking the button a new window opens and lists the referred software that seems to be missing. It offers the possibility to tidy up and delete the allocation (from the AUDIT_SOFTWARE_TO_LICENSE_POOL table).

**Navigating the Software Table**

The navigation of the software table deviates from the usual since selecting an item in the software table entails a change in item allocation. Therefore the location of the cursor in the table is not illustrated by a highlighted row but as a red star at the beginning of the row.
The star, meaning the row cursor, changes position by

- an implicit search of a table entry (e.g. automatically when showing the assigned software items when the license-pool entry in the license-pools table above is changed),
- a manual search using the search bar,
- using the red arrow buttons located directly above the table,
- clicking directly into the assigned column for the star (the very first column) of the desired row (while clicking any other column of that row resets the entire selection).

The allocation of software items to license-pools as a basis for a compliance check

The main principle is simple: When a software pool is established, the utilized software items for this pool are selected with a few mouse clicks. This forms the data basis (for example for the compliance check from Microsoft) to determine the number of installations linked to a pool. The result of this evaluation can be found in the Statistics tab (Section 9.10.8, “License usages overview”).

The allocation of software items to license-pools has to be maintained meticulously to receive correct results. Especially when software from a license-pool is installed, the specific software item has to be registered to the respective pool.

The different possible configurations of the displays are used to provide functions to support various policies and best practice in data maintenance.

Function: Filtering the assigned software items

The context menu of the table provides the option to switch between viewing all entries (including the highlighted rows) or exclusively the marked ones. Thereby one obtains a list of all assigned software items in the default configuration. This can be helpful since the rows of assigned software items do not necessarily follow each other directly in the overall software list.
Instead of using the context menu, one can also click on the filter icon next to the search bar which gives the same result. When all rows are shown this icon appears as an open filter, and when only the highlighted rows are shown the icon appears as a closed filter.

In the filtered view the allocations can be edited as well. In this case, allocated items can only be removed.

**Function: Completing the allocations to a software name**

When a manufacturer of a software releases a new version or patch, the new version receives a new windowsSoftwareId and therefore shows up as a new entry in the software table (with a new ID). In most cases, the new software can be treated the same way for licensing as the previous version. If a license agreement was required for the old version, the same likely holds for the new one. This means the demand for a license has to be registered via the same license-pool for both versions. Covering the demand can then be organized via the shared license-pool.

It is important to make sure when rolling out a new version that an allocation is added to the respective license-pool. Besides, some software updates itself without active involvement. Then after the update, the correct allocation might be missing maybe even unnoticed. That is why (since version 4.1.9.8) the opsi-configed includes an auxiliary function to simplify the completion of allocations.

As a basis for this we assume that the data field name is often sufficient to identify the license-pool, and differences in components of the software ID can (often) be ignored in this regard. The functions and options in the gray box with the label Alternative view serve for working with this hypothesis. The button Name → Pool opens an additional window.
The additional window contains two auxiliary tables. When prior to calling the window the default setting was left at *Show all software names*, the first column of the first table now displays the software names in alphabetical order. The second column lists all IDs that are associated with the respective name (although the first part of the ID, being the exact software name, is not repeated for simplicity reasons).

When selecting a name in the first table, the dependent second table displays the corresponding IDs individually and specifies if an allocation to a pool exists, and if so to which pool. The *license-pool* allocations can be unified across all rows of the table in three ways:

- Remove all allocations,
- Allocate the currently (in the main window of the license management) selected pool to all ID variations (this is presumably the standard case),
- When the different variations are currently assigned to different pools, assign all of them to the pool belonging to the selected row.

Since one would like to systematically check the *license-pool* allocations, it is recommended to click the option to *show software names without uniform assignments* prior to calling the window. When this option is set, the upper table is reduced to the software names that correspond to multiple IDs where those IDs are also assigned to different *license-pools*.

If such allocation differences exist, an "i" in a small Box within the gray Box of the license management appears as an indicator. This information is designed discreetly, since, depending on the licensing case, it can be correct or even needed to have different software variations licensed differently.
To search for possible forgotten license allocations, the additional window can be opened with the third and final option *show software names without any assignment*.

**Function: Recognizing non allocated Software**

Instead of relying on the previously depicted process, thorough license management can also be achieved by assigning *every* registered software to a *license-pool*.

For this to work in practice, pseudo *license-pools* like "free Software" or "Operating System components" are needed to which software items that do not have or require any separate licensing can be assigned to.

To simplify the implementation of this strategy, the *opsi-configed* offers this additional display option (cf. Figure 104, “Software table configuration”):

![Figure 110. Options for displaying software items](image)

Assuming the initial situation is that the previously registered software has universally been assigned to *license-pools*, then the following sequence of steps are expedient:

1. Installing a (new) software on a computer.
2. Running *swaudit* on the computer.
3. If the software is indeed new, the software table receives a new entry; there can also be multiple entries (for example when libraries are installed).
4. When preselecting a suitable *license-pool* and activating the option *Show software names without uniform assignments*, then the new table entries are the only highlighted ones.
5. If the preselected *license-pool* is suitable, a new entry for this *license-pool* can be added to the previously registered with "Ctrl + click".

For the distribution of arbitrary, initially not determined software items to *license-pools*, the following procedure can be used:

1. Activate the option *Show software names without any assignments*
2. Change to edit mode *Selection = SW items to assign (additionally)*
3. Select and highlight rows using "Ctrl + click"
4. Select the *license-pool* that the highlighted software items should be assigned to. The save button is now active.
5. Save the changes

6. Reset the edit mode to **Selection = Complete list of SW items assigned/to assign**

Please note that when the table filter is active, the functions described here act differently.

### 9.10.4. Setting up licenses

Setting up a license or supplying a license to a **license-pool** requires several steps. These can be performed in the **New license** tab, the second tab in the license management window.

At the top there is a table of available **license-pools** to select the **license-pool** that the new license is to be assigned to.

![License management: "New license" tab from the license management window](image)

**Figure 111. License management: "New license" tab from the license management window**

Before continuing with the next steps, some basic concepts and terms of license management have to be introduced:
Some aspects and terms of the license concept

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

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

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

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

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

**Registering the license contract**

The next step after selecting the license-pool for the new license option is to register the license contract that the license should be based on. Under "Select or enter license contract" (in the "New license" tab) an existing contract can be selected or a new one can be created.

The license contract dataset comes with the data fields partner, conclusion date, notification date, and expiration date. The entry field notes can hold some additional information like the location where the contract document is kept. The unique contract ID (licenseContractId) is for identifying the license contract in the license management database. When entering a new license contract, a new unique ID is automatically constructed based on the current date and time stamp. The default setting can be used if the license contract is implied by purchasing the software or the contract is documented and traceable some other way. Otherwise, the data can and should be edited to ensure orderly tracking of the underlying contract which can be for example through reference to a file number in the notes field.

The contract ID can only be changed before saving the new data set. When saving the data, the opsi service checks whether the ID is unique. In case it is not, a new ID is generated and cannot be changed anymore.

**Configuring the license model**

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

Several types of license models are available:

- Standard license
• Volume license
• OEM license
• Concurrent license

Each option is represented by a button. By clicking one of the buttons the form is filled with data for that type of license model.

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

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

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

The **Concurrent license** means that a certain number of licenses is available for a variable set of clients. Within opsi, this situation is handled like an unlimited Volume license. The number of actual installations in use has to be managed by some external license server.

After clicking a button, the automatically generated data include a unique generated ID (derived from date and time stamp). This ID can be changed as desired.

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

The field "Expiration date" defines the expiration date of the license in a technical sense. (This column of the license is for future use).

**Saving the data**

The "Send" button sends the data to the opsi service to save them permanently to the opsi database (if they are consistent and no errors occur).

While proceeding with this, data records will be generated for the new software license based on the selected software contract and the new license option assigned to that.

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

**9.10.5. Editing licenses**

In ninety percent of the use cases editing the license data with help of the tabs "License pools" and "New license" will do. But there might be some special cases where more specific and explicit editing of the license data is needed. For this, the "Edit licenses" tab presents the license data in three tables, representing the internal data structure and allowing to adapt the data for some special cases.
Based on this direct data access, the following chapter shows how to configure a special license, like the Microsoft Vista or Windows 7 Professional downgrade option for installing Windows XP.

Example downgrade option

The downgrade option means, that instead of the purchased software, the preceding version can also be installed. For instance, installing Windows XP based on a Windows Vista license. In this case, the license key can also be used for an installation, which it was not meant for originally.

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

From the "New license" tab the Vista license is to be registered, as usual, resulting in a new license option, which is displayed in the list of license options at the bottom of the window. This new license option is based on a new software license identified by `softwareLicenseId`.
This `softwareLicenseId` is needed for further configuration steps. You can keep it in mind or copy it with drag&drop. You can look for the ID in the "Available license options" list of the "Edit licenses" tab as well. The context menu also supports copying the ID.

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

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

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

**9.10.6. Assignment and release of licenses**

Using a license option by installing the software on a client results in the actual licensing.
In the opsi context, installations are done script-based and automatically. The client running the Winst script then invokes calls to the central opsi service.

The following chapters introduce some of these service calls, which are relevant for license management. For further information about Winst and opsi commands see the documentation on Winst and opsi.

**opsi service calls for requesting and releasing a license**

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

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

The use of a license option can be released by calling `deleteSoftwareLicenseUsage`.

Again the parameters to be passed are the `hostId` and alternatively the `licensePoolId`, `productId` or Windows Software ID. Calling this method releases the license option and returns it to the pool of available license options.

For the complete documentation of opsi service calls see below.

**opsi-winst script calls for requesting and releasing of licenses**

The `opsi-winst` provides the client-related calls as `opsi-winst` commands.

An `opsi-winst` script can make a call to the function `DemandLicenseKey` to get a license key for installing. The parameters to be passed are:

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

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

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

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

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

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

The boolean function `opsiLicenseManagementEnabled` can be used to check whether the opsi license management is enabled and can be used for scripting:
if opsiLicenseManagementEnabled
    set $mykey$ = DemandLicenseKey ("pool_office2007")
else
    set $mykey$ = IniVar("productkey")

The service calls can be invoked from the command-line tool **opsi-admin**.

Parameters marked with * are optional.

### License contracts

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

This method registers a new license contract record with the ID `licenseContractId`. If no `licenseContractId` is passed, it will be generated automatically. Using the `licenseContractId` of an existing contract, this contract can be edited.

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

The method returns the `licenseContractId` of the contract.

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

With the string returning functions `getLastServiceErrorClass` and `getLastServiceErrorMessage` error states can be detected and handled, e.g. if there is no license available:

```plaintext
if getLastServiceErrorClass = "None"
    comment "no error"
endif
```

The error class `LicenseMissingError` is returned if a license has been demanded but there is no license available. The error class `LicenseConfigurationError` is returned if the current configuration does not allow assignment of a license pool to a software. This could be the case if either no assignment exists or no distinct assignment is possible.
Manual administration of license usage

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

![Image: License management: "License usages" tab from the license management window]

These are the functions for manual license management in detail:

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

Preservation and deletion of license usages

If a software packet is reinstalled, the call to the `opsi-winst` function DemandLicenseKey will return the same license option and license key as used before.
In case this is not favored, the former license option has to be released by calling the `opsi-winst` command `Freelicense`, or by calling the opsi service call `deleteSoftwareLicenseUsage`, or deleting the license use manually.

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

To release the licenses, they can be deleted from the "License usage" tab or deleted by the service call `deleteAllSoftwareLicenseUsages` by passing the client host name as a parameter.

### 9.10.7. Reconciliation with the software inventory

The tab "Reconciliation" lists for each client and each license-pool whether the use of this license-pool is registered by opsi ("used_by_opsi") and if the software inventory (`swaudit`) on that client reported a software that requires a license option from that pool (`Swinventory_used`).

To evaluate the results from `swaudit`, the relevant software IDs (as found in the client registry) have to be associated with the appropriate license-pool (tab "License pools").

When matching the data with the software inventory, the license management counts not more than one license per client and license-pool. So for example if the license-pool `office2010` is connected with ten different patterns from software inventory, indicating that `office2010` is installed on this client, this is (regarding the license usages count) counted as a single installation, although all of the detection patterns might be found on the client.

![Figure 115. License management: "Reconciliation" (data matching) tab with the inventory](image-url)

As usual, this table can be copied using Drag & Drop and for instance pasted to a spreadsheet program. If the `opsi-configed` process has the required access rights (running standalone and not from...
By virtue of the config `config.license_inventory_extradisplayfields` which can be edited in the host parameter page of the server, you may add extra data fields for each client to the table.

### 9.10.8. License usages overview

The tab "Statistics" displays a summary of the different license-pools, showing the total number of license options (`license_options`) and how many of them are in use (`used_by_opsi`) or still available (`remaining_opsi`).

![License management: "Statistics" tab from the license management window](image)

In addition to the number of license-uses registered by opsi (`used_by_opsi`) and the currently available licenses (`remaining...`), the overview also shows the total number of detected installations, that require a license (`SWinventory_used`).

The data from the column `SWinventory_used` is based on the registry scans from the `opsi-product swaudit` and the assignment of the Windows software IDs (as they are found in the registry) to the license-pools (as registered with the opsi license management (tab "License pools", see Section 9.10.3, “license-pools”).

From the context menu, the table can be printed (because of restricted access rights not available from the applet), and using drag&drop data can be copied to e.g. a spreadsheet.

#### In case of downgrade option

If a downgrade option has been configured (see Section 9.10.5.1, “Example downgrade option”), it
A single downgrade license results in a license option for at least two different license-pools but only one of them can be requested for an installation. So using a downgrade license option decreases the number of available license options \( \textit{remaining\_opsi} \) in each of the license-pools concerned by that downgrade option by \( 1 \). So it looks like a single installation reduces the number of available license options by \( 2 \), which, in this case, actually is the fact.

### 9.10.9. Service methods for license management

The service methods for license management can be called from the command-line tool \textit{opsi-admin}, so they are accessible for scripting, e.g. to read license keys from a file.

Examples can be found in the products \textit{license-test-....opsi} from https://download.uib.de/opsi4.1/misc/license-management/. After installing the packages with \textit{opsi-package-manager -i *opsi}, in the directory /var/lib/opsi/depot/<product name> the corresponding scripts: create_license-*\_sh can be found.

As an example here the script \textit{create\_license\_mixed.sh} (the current version comes with the download packet).

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

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

#############
createlic ()
{
    while [ -n "$1" ]
do
        #echo $1
        AKTKEY=$1
        shift
        #echo $1
        AKTHOST=$1
        shift
        echo "createSoftwareLicense with oem key: $\{PRODUCT_ID\}-oem-$\{AKTKEY\} for host $\{AKTHOST\}"
```

9. opsi Extensions
MYLIC="opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID" "RETAIL" "1" "${AKTHOST}" """
    opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC" "p_$PRODUCT_ID" "$PRODUCT_ID-retail-${AKTKEY}"
done
}

# here the script starts

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

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

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

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

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

# create the retail licenses:
for AKTKEY in $MYRETAILKEYS
do  
    echo "createSoftwareLicense with retail key: ${PRODUCT_ID}-retail-${AKTKEY}"  
    MYLIC="opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID" "RETAIL" "1" "${AKTHOST}" """
    opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC" "p_$PRODUCT_ID" "$PRODUCT_ID-retail-${AKTKEY}"
done
9. opsi Extensions

9.10.10. Example products and templates

In the uib download section at [https://download.uib.de/opsi4.1/misc/license-management/](https://download.uib.de/opsi4.1/misc/license-management/) are four example products available. One for each type of license model, as there are Retail, OEM and Volume license type, as well as a product combining all of them.

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

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

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

9.11. opsi WAN/VPN extension

The WAN/VPN extension module allows to integrate clients, that are connected via low speed connections. This chapter is about configuring and maintaining the opsi WAN/VPN extension.

9.11.1. Preconditions for using the WAN/VPN extension

This opsi extension is currently in the cofunding process and not free. For more details see Section 9.1, “Activation of non-free modules”.

There are some preconditions to use the WAN/VPN extension module. The feature product groups is required, which is available with opsi 4.0 and above. Also the packets opsi-client-agent and opsi-configed are required, which come with version 4.0.1.

At the moment, the simultaneous use of both “WAN extension” and “installation on shutdown extension” is not supported. On the same opsi server with different clients, these opsi extensions can be used.

*Table 23. Required packets*
<table>
<thead>
<tr>
<th>opsi-Packet</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-client-agent</td>
<td>&gt;=4.0.1-1</td>
</tr>
<tr>
<td>opsi-winst</td>
<td>&gt;=4.10.8.12</td>
</tr>
<tr>
<td>python-opsi</td>
<td>&gt;=4.0.1-7</td>
</tr>
<tr>
<td>opsi-configed</td>
<td>&gt;=4.0.1.6-1</td>
</tr>
</tbody>
</table>

9.11.2. General overview of the WAN/VPN extension

.opsi software deployment is mainly doing the following steps:

- The `opsi-login-blocker` at client system startup prevents the users from logging on.
- The `opsiclientd` service running on the client connects the `opsi-config-server`.
- If any product actions are set for the client, it mounts a share from the `opsi-depot`.
- The `opsi-winst` is starting and also connects to the `opsi-config-server`.
- The `opsi-winst` executes the product actions, using the share from the `opsi-depot`.
- If a reboot is required, it executes and the process starts all over.
- When all the product actions are completed, the log files are transferred to the `opsi-config-server` and the user logon is unblocked.

Now we will look at the special circumstances of a client, which is located in a remote branch, connected via WAN to the LAN, where the `opsi-config-server` and `opsi-depot-server` are:

- During communication with the `opsi-config-server` small amounts of data are transferred, so there is no noticeable slowdown of the software deployment in a WAN.
- But processing the product actions might consume a very long time, depending on the packet sizes, bandwidth and latency of the WAN connection. There also might occur timeouts during file access.
- Therefore, during the installation is processing, the user has to wait for an unreasonably long time before logon is granted.

As an alternative to providing a dedicated `opsi-depot-server` in the remote branch network, the remote clients can be connected via WAN/VPN extension module. Using the WAN/VPN extension module, the `opsi-client-agent` can be configured this way:

- At system startup, if there are no opsi-products cached and ready for installation, the user can logon immediately.
- When there are product actions set for the client, the `opsiclientd`, which is running on the client, starts downloading the required installation files from the `opsi-depot` to the local file system. This is done in the background while the user is working. The maximum download bandwidth can be configured and also can be dynamically adapted to the current network traffic status.
- When the synchronization of the `opsi-products` is completed, a reboot request is triggered.
- The logged on user can accept the reboot request, or the client will be rebooted at some time later...
• At the next system startup, the cache is found to be filled with the \textit{opsi-products} to be installed and the installation starts. In this case, the installation will use the downloaded files from the local file system, which increases the speed of installation even compared to a standard \textit{LAN} installation.

Now we examine the situation of a notebook, which at system startup often cannot connect the \textit{opsi-config-server}:

• Trying to connect the \textit{opsi-config-server} at system startup will result in a timeout.
• Connecting the \textit{opsi-config-server} might be possible when a user logs on and a \textit{VPN} connection to the corporate network is established.
• Without connection the \textit{opsi-config-server} no software deployment is available.

This situation also can be solved by using the WAN/VPN extension module. The \textit{opsi-client-agent} can be configured the following way:

• At system startup, if there are no \textit{opsi-products} cached and ready for installation, the user can logon immediately.
• Triggered by network activation or a by timer event, the \textit{opsiclientd} running on the client tries to connect the \textit{opsi-config-server}.
• If the \textit{opsi-config-server} is reachable, the \textit{opsiclientd} starts to:
  ◦ synchronize the configurations
  ◦ download the required files from the \textit{opsi-depot} to the local file system.
    In combination with the opsi extension module \textit{Dynamic Depot Selection}, the download is done from the best fitting \textit{opsi-depot}.
• When the synchronization of the \textit{opsi-products} is completed, a reboot request is triggered.
• The logged on user can accept the reboot request or the client will be rebooted at some time later on.
• At the next system startup, the cache is found to be filled with the \textit{opsi-products} to be installed and the installation starts. In this case, the installation will use the downloaded files from the local file system, which increases the speed of installation even compared to a standard \textit{LAN} installation. So the \textit{opsiclientd} takes over the function of both the \textit{opsi-config-server} and the \textit{opsi-depot-server}.
• At the next connect to the \textit{opsi-config-server} the results of the installation process (configuration change, log files ...) will be synchronized.

The download mechanism of product synchronization is multiple interruptible and will continue at the point of interruption. So files that are already downloaded will not have to be downloaded again.

The WAN/VPN extension module allows to connect clients, that are outside of the secure corporate network. Therefore additional security mechanisms are required regarding the communication between client and server.
So the \textit{opsiclientd} now offers the ability to verify the identity of an \textit{opsi-server}. Therefore the key pair of the SSL certificate of the \textit{opsiconfd} is used.
By this mechanism the opsiconfd-server as well as the opsidepot-server can be verified, assumed the communication is performed via opsiconfd and SSL. In case of an opsidepot the file access must be done by the opsiconfd using HTTPS/WEBDAVS. Access done via CIFS/SMB will not be checked.

9.11.3. Caching of opsi-products

Caching of opsi-products is done by the ProductCacheService, which is part of the opsiclientd. The ProductCacheService synchronizes the local copy of an opsi-product with the current version of the corresponding opsi-products on the opsidepot. The location of the local product cache can be configured and defaults to %SystemDrive%\opsi.org\cache\depot.

Communication Protocol for accessing an opsi-depot

For transferring the product files, two different protocols are used:

- CIFS/SMB
- HTTP(S)/WEBDAVS(S)

In case of using CIFS/SMB, a connection to the depotRemoteUrl will be established as configured with the properties of the opsi-depot. In case of using HTTP(S)/WEBDAV(S), the depotWebdavUrl is connected, which as well is to be configured with the properties of the opsi-depot.

Which protocol is to be used, can be configured client specific by the host parameter clientconfig.depot.protocol. Available values to be set as clientconfig.depot.protocol are cifs and webdav.

Also the opsi-linux-bootimage is evaluating this setting and uses the specified protocol.

Using the .files file for Synchronization

The synchronization process is based on the file <product-id>.files, which is located in the base directory of each opsi-product on the opsi-depot. This file contains information about the files, directories ans symbolic links used by an opsi-product. Each line of that file contains such information. Different types of information are separated by a blank. The first character of a line defines the type of the following entry. Available values are:

- d for a directory
- f for a file
- l for a symbolic link

Separated by a blank follows the relative path, which is single quoted. The next entry gives the sizes of the file (which is 0 for directories and symbolic links). The final entry in case of a file is the MD5-sum of the file, in case of a symbolic link it is the target referred to by the symbolic link.
Example excerpt of a `.files` file:

```plaintext
d 'utils' 0
f 'utils/patch_config_file.py' 2506 d3007628addf6d9f688eb4c2e219dc18
l 'utils/link_to_patch_config_file.py' 0 '/utils/patch_config_file.py'
```

The `.files` file is generated automatically when installing `opsi-product-packages` (after running the postinst-Script).

> When using the WAN/VPN extension, the files of `opsi-products` on the `opsi-depot` should not be changed manually, otherwise the information contained in the `.files` file would be outdated, causing errors during the synchronization process.

### Internal processing of opsi-product caching

The synchronization of a local copy of an `opsi-product` processes as follows:

- The `.files` file of the `opsi-product` is transferred to the local client.
- Then it is checked, whether there is enough local disk space available to cache the `opsi-products`. If there isn't enough disc space available, some old `opsi-products` will be deleted, which haven't been used (synchronized) for a long time.
- The local caching directory will be created if it doesn’t exist.
- Referring to the `.files` file, any old files and directories, which aren’t in use anymore, will be deleted from the local `opsi-product` cache.
- Then the `.files` file will be processed in the following order.
  - missing directories are created.
  - missing files are transferred.
  - existing files will be checked by size and MD5-sum and be synchronized again if necessary.

The synchronization results in an exact local copy of the `opsi-product` from the `opsi-depot`.

> On windows systems, no symbolic links will be created. Instead of links there will be copies of the link target.

When the `opsi-product` has completed successfully,

- the status of `productsCached` will turn to `true` (and stays `true` in case of a system restart, see: Section 6.1.3.7, “Configuration of different events”).
- a `sync completed` event will be triggered.

### Configuring the opsi-product caching

The `opsi-product` caching is configured in the section `[cache_service]` of the `opsiclientd.conf`. 
• **product_cache_max_size** (integer): The maximum size of the *opsi-product* cache in byte. This is important to limit the disk space to be used by *opsi-product* caching.

• **storage_dir** (string): the path to the directory, in which the base directory *depot* for the *opsi-product* caching is to be created.

Further configurations can be done event specific.

Within an event configuration section `[event_<event-config-id>]` the following options are available:

• **cache_products** (boolean): if the value of this option is **true**, in case of the event the *ProductCacheService* will start to cache *opsi-products*, for which a *product action* is set. If additionally the value of the option **use_cached_products** is set to **true**, the further processing of this section will be suspended until the caching of *opsi-products* is completed.

• **cache_max_bandwidth** (integer): the maximum bandwidth in byte/s to be used for caching. If this value is set to 0 or less, the bandwidth is unlimited.

• **cache_dynamic_bandwidth** (boolean): if the value of this option is set to **true**, the bandwidth will be adapted dynamically. Therefore the network traffic at the network interface to the *opsi-depot* will be monitored. If any traffic is detected, which is not caused by the *opsi-product* caching, the bandwidth for the caching will be sharply reduced, to allow other processes to work at (almost) full speed. If the caching works at reduced bandwidth and no more other network activity but the *opsi-product* caching is detected, the caching process will continue with unlimited bandwidth. The value of **cache_max_bandwidth** will be used to limit the maximum dynamic bandwidth.

• **use_cached_products** (boolean): if the value of this option is set to **true**, the local *opsi-product* cache will be used for processing *product actions*. If caching of the *opsi-products* is not completed, the processing of this event will stop and return an error code.

### 9.11.4. Caching of configurations

The caching of configurations is done by the *ConfigCacheService*, which is part of the *opsiclientd*. The *ConfigCacheService* synchronizes a local *client-cache-backend* with the *config backend* of the *opsi-config-server*.

The *opsiclientd* provides per *WebService* an access point to the backend and thereby provides quite the same functionality as the *opsiconfd*.

#### The local client-cache-backend

The local *client-cache-backend* is based on *SQLite* and mainly consists of a local working copy, a snapshot an a modification tracker, which records all changes of the local working copy.

The base directory of the config cache can be configured and defaults to `%SystemDrive%\opsi.org\cache\config`. The snapshot reflects the configuration status on the *opsi-config-server* at the time of the last synchronization.

At the start of the processing, the local working copy is a copy of the snapshot, but will be modified during processing.
**Internal processing of configuration synchronizing**

The synchronization of the local changes of the `client-cache-backend` with the `config backend` of the `opsi-config-server` is processed as follows:

- The changes of the local working copy registered by the modification-tracker are transferred to the `opsi-config-server`. Any changes of the configuration on the `opsi-config-server` since the last synchronization will be detected by comparing to the snapshot. If there are any conflicts detected, the following rules apply:
  - In case of inventory data, the local client data have priority.
  - In case of `action requests`, the value from the `opsi-config-server` has priority.
  - In case of `installation status` and `action result`, the client data have priority.
  - If the opsi license management module is switched on (config: `license-management.use=true`), the config server tries to find a license pool for the product by the assignment pool to `productId`. A free license of this pool will be reserved and this license will be replicated. Any unused licences, which have been reserved during replication, will be released again.
  - The `opsi-config-server` has priority for the status of `host parameter` and `product properties`.
- The modification tracker will be cleared.
- The logfiles will be transferred.

The `config backend` replication of the `opsi-config-server` to the `client-cache-backend` is processed as follows:

- The replication only takes place, if any `action requests` are set on the `opsi-config-server`. The `product action always` does not count in this respect. The replication process will start only if the status of `action requests` is changed since the last replication.
- The modification tracker and the local working copy are cleared.
- The configurations required for local processing will be replicated.
- If `action requests` are set for `opsi-products` which are marked as to require a license, the required software license will be reserved from a `license pool`, which is assigned to that `opsi-product`.
- Additionally required data, as there are the `auditHardwareConfig` and the `modules`, will be transferred.
- The snapshot and the local working copy will be updated, so they have the same content.

A successful replication from server to client results in:

- The status of `config_cached` is set to `true` (and stays `true` in case of a system restart, see: Section 6.1.3.7, “Configuration of different events”).
- An event of type `sync completed` will be triggered.
Configuration of config caching

The configuration of the config caching is mainly done event specific:

Within an event configuration section [event_<event-config-id>], the following options are available:

- `sync_config_to_server` (boolean): if the value of this option is set to `true`, the `ConfigCacheService` in case of that event starts to transfer the changes registered by the modification tracker to the `opsi-config-server`. The process will wait for that task to complete.

- `sync_config_from_server` (boolean): if this value is set to `true`, the `ConfigCacheService` starts with the replication. If additionally the value of the option `use_cached_config` is set to `true`, the processing of this event is suspended until the replication is completed.

- `use_cached_config` (boolean): if the value of this option is set to `true`, the `client-cache-backend` will be used for processing the `product actions`. If the synchronization is not completed, the processing of this event will be stopped and return an error code.

- `post_sync_config_to_server` (boolean): has the same functionality as `sync_config_to_server`, but will be evaluated after the `product actions` have been completed.

- `post_sync_config_from_server` (boolean): has the same functionality as `sync_config_from_server`, but will be evaluated after the `product actions` have been completed.

9.11.5. Recommended configuration when using the WAN/VPN extension module

The `opsi-client-agent`-package comes with a `opsiclientd.conf` prepared for the WAN/VPN extension. For activating the WAN/VPN extension, just enabling of some events and disabling of some others is required.

The configuration of the `opsi-client-agent` also can be done from the web service (see: Section 6.1.3.12, “Configuration via web service (Host Parameter)”), so it is recommended to create the following `host parameter`:

- `opsiclientd.event_gui_startup.active` (boolean, default: `true`)
- `opsiclientd.event_gui_startup{user_logged_in}.active` (boolean, default: `true`)
- `opsiclientd.event_net_connection.active` (boolean, default: `false`)
- `opsiclientd.event_timer.active` (boolean, default: `false`)

By these `host parameter`, events can be enabled or disabled client specific. The `host parameter` can be created using the `opsi-configed` or `opsi-admin`.

For creating the `host parameter` by `opsi-admin`, the following commands have to be executed on the `opsi-config-server`:
The default values are as they come with the special `opsiclientd.conf`.

If you do **not** set the defaults like described above and skip directly to the commands below you set **all** your clients into WAN mode!

For a WAN/VPN client, which shall do caching of configurations and *opsi-products*, the *host parameter* have to be configured as follows:

- `opsiclientd.event_gui_startup.active: false`
- `opsiclientd.event_gui_startup{user_logged_in}.active: false`
- `opsiclientd.event_net_connection.active: true`
- `opsiclientd.event_timer.active: true`

The client specific *host parameter* can be set by `opsi-configed` or `opsi-admin`.

To set the *host parameter* by `opsi-admin`, the following commands have to be executed on the `opsi-config-server`: (in this example the client has the `opsi-host-Id` `vpnclient.domain.de`):

```sh
opsi-admin -d method configState_create opsiclientd.event_gui_startup.active vpnclient.domain.de false
opsi-admin -d method configState_create opsiclientd.event_gui_startup{user_logged_in}.active vpnclient.domain.de false
opsi-admin -d method configState_create opsiclientd.event_net_connection.active vpnclient.domain.de true
opsi-admin -d method configState_create opsiclientd.event_timer.active vpnclient.domain.de true
```

This configuration will process as follows:

- At system start of the client there will be no connection established to the `opsi-config-server`.
- When the activation of a network interface is detected, a connection to the `opsi-config-server` will be established (if possible) and the synchronization starts as background task.
- A *timer*-Event will be established, which tries at regular intervals to trigger the synchronization.
Setting the protocol for caching of opsi-products

The caching of opsi-products can be done via the protocols HTTPS/WEBDAVS or CIFS/SMB.

When using webdav, access to the opsi-depot is performed by the opsisconfd.

- advantages:
  - easy firewall configuration, for it requires just port 4447.
  - verify of the SSL-certificate of the opsi-depot available.

- disadvantage:
  - the opsisconfd generates more traffic on the opsi-depot.

By using cifs, access to the opsi-depot is done via SAMBA.

- advantage:
  - the SAMBA-server shows a good performance, is resource-conserving and well scaleable.

- disadvantages:
  - the firewall configuration is more complicated, access to the SAMBA ports is required.
  - no verify of the SSL-certificate of the opsi-depot is available.

An instruction for configuring the protocols is to be found in the chapter Section 9.11.3.1, “Communication Protocol for accessing an opsi-depot”.

Verifying the server certificates

To activate the verifying of SSL certificates, in the opsiclientd.conf within the section [global], the option verify_server_cert is to be set to true. This, during connection to an opsisconfd, results in
verifying the opsiever by using the SSL certificate. The server certificates will be stored in the directory %SystemDrive%\opsi.org\opsiclientd\server-certs. The name of the certificate is combined from the server address (IP or name) and the extension .pem. If at connection time no stored certificate is to be found, no checking will be done.

To publish a changed certificate, the old certificate stored on the clients has to be deleted. This can be done by the RPC-method deleteServerCerts, which is available from the control interface of the opsiclientd.

9.12. FAQ for setup and application of the WAN extension

9.12.1. Do I need a license?

Yes, the extension is still part of the cofinancing scheme and therefore subject to a fee. Additionally for the operation of the WAN extension the MySQL backend is required which is also not free. You can test the extensions free of charge for a limited amount of time. In order to do so please contact info@uib.de

9.12.2. How do I activate the WAN extension on a client?

The activation of the WAN extension can be performed per client via the checkbox on the management console. The client will receive the specific configuration change the next time it connects to the opsi server. It then becomes active the next time the client is restarted.

9.12.3. Must the opsi client agent be deployed once again for the activation?

No, that is not necessary since the client can get the configuration automatically at the next connection to the opsi server. That can be at the next boot process or after an "on_demand" event.

9.12.4. How can I activate clients that are already in Home Office?

If the clients can be reached by the opsi server in home office, the change of the configuration can be performed by triggering an "on_demand" event. (This only works if the data transfer is set to cifs)

In the event that the clients in the home office cannot be reached by the opsi server, they can be updated via this opsi-admin call. (This should also work for clients that have been switched to webdav)

On the command line of the server this can be done using:
Replace the corresponding entries:

1. `<ClientIp>` = IP address of the clients
2. `<user>` = local administrator on the client or a member of the local administrator group
3. `<pass>` = password of the administrator
4. `<FQDN>` = FQDN of the client
5. `<OpsiHostKey>` = opsi hostkey of the client

9.12.5. How big is the cache on the client?

The cache on the client is set to 20 GB by default. The programs and updates to be installed are stored here. During an update, only the changed data is copied to the cache directory via a sync process. The size of the cache should be chosen generously so that large packages can also be saved.

9.12.6. How can I change the size of the cache on the client?

With this command, this parameter will be created in the management console. The value can then be individually adjusted for each client:

```bash
opsi-admin -d method config_createUnicode
"opsiclientd.cache_service.product_cache_max_size" "Maximum product cache size in bytes"
"40000000000" "40000000000" "True" "False"
```

9.12.7. How can I increase the time the client waits for a response before rebooting for an installation?

With this command, this parameter will be created in the management console. The value can then be individually adjusted for each client:

```bash
opsi-admin -d method config_createUnicode
"opsiclientd.event_sync_completed{cache_ready_user_logged_in}.shutdown_warning_time" 
"How long should to wait for an answer before a reboot. The default is 3600 seconds.
"3600" "3600" "True" "False"
```
If you want to increase the time by 8 hours enter 28000 seconds here.

**9.12.8. "on_demand" installations in WAN mode?**

The installations are triggered as if the client would not be a WAN client.

1. The config server receives a direct request.
2. A Samba share is mounted via cifs. If webdav has been activated for data transfer, this will be ignored.
3. The installation starts immediately without the data being cached beforehand.

**9.12.9. Does the kiosk mode work in WAN mode?**

The kiosk mode works only if the opsi config server is available. Also in WAN-mode the kiosk connects directly to the opsi server. If the request to install a package is not only saved but also executed directly, the installation is performed the standard way and not in WAN mode, which means the packages are not cached. You can use the kiosk on a WAN client, but the actual WAN mode components are not used.

**9.12.10. Can the Install by shutdown mode be combined with the WAN mode?**

No, Install by shutdown mode is not compatible with WAN extension.

**9.12.11. Subscription MSHotfix and WAN mode**

Basically you can deploy the Mshotfix packages from your subscription on your WAN clients. However, it should be noted that these standard packages are very large in size (e.g. Windows 10 64 bits is around 6GB).

For this reason "modular" Mshotfix packages are provided for Windows 10 that only contain the patches for a certain ReleaseID, e.g. mshotfix-win10-1809-x64-glb. The size of these "modular" packages is currently about 1 GB). Therefore we strongly suggest to use these "modular" packages in WAN mode.

**9.12.12. How can I clear the cache on a client?**

In configed on the clients-tab via the OpsiClient-menu "On WAN-clients: Delete package caches", or

```
opsi-admin -a https://<ClientIp>:4441/opsiclientd -u "<FQDN of the client>" -p "OpsiHostKey" method "cacheService_deleteCache"
```

**9.12.13. Can user-login-scripts be used in WAN mode?**

No, that makes no sense because these are started when the user logs in to make changes to roaming profiles that cannot be used with WAN clients. The entry in opsi-configed for
opsiclientd.event_user_login.active should therefore be set to false.

9.13. opsi-Nagios-Connector

9.13.1. Introduction

Beside client management is monitoring one the central functions in a modern IT service management. With opsi you got a client management tool. For monitoring tasks there are other well known open source solutions. So we build for the monitoring tasks in opsi not an own monitoring tool but an interface to existing solutions. With this opsi extension we provide a connector to Nagios. In the following chapters the design and configuration of the opsi-Nagios-Connector is explained.

The opsi-Nagios-Connector isn’t strictly bound to Nagios. It is developed for the use with Nagios and Icinga. It should also work with other Nagios derivatives but this is whether tested nor supported.

The scope of this manual is the design and configuration of the opsi-Nagios-Connector. It is not a Nagios manual. You will need a running Nagios installation and the knowledge how to run Nagios.

9.13.2. Preconditions

**Preconditions at the opsi server and client**

This extension is at the moment a cofunding project which means that until the complete development costs are payed by co-funders, they are only allowed to use by the co-funders or for evaluation purposes. If we have earned the development cost we will give these modules for everybody for free.

see also

opsi cofunding projects
opsi cofunding contribution

So as precondition to use this extension you need as first an activation file. For evaluation purpose you will get a temporary activation file if you ask for it in a mail at info@uib.de.

For more details see Section 9.1, “Activation of non-free modules”.

Technical preconditions are opsi 4.0.2 with the following package and product versions:

<table>
<thead>
<tr>
<th>opsi package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-client-agent</td>
<td>&gt;=4.0.2-1</td>
</tr>
<tr>
<td>opsiconfd</td>
<td>&gt;=4.0.2.1-1</td>
</tr>
<tr>
<td>python-opsi</td>
<td>&gt;=4.0.2.1-1</td>
</tr>
</tbody>
</table>

**Preconditions at the Nagios server**

As precondition you need a Nagios installation in the version 3.x or a Icinga Installation in the version
1.6 or higher.
For graphical output of performance data a pnp4nagios installation is required.

Further information can be found at:

• nagios.org
• icinga.org
• pnp4nagios.org

9.13.3. Concept
The opsi-Nagios-Connector contains of two core components. At first we will discuss these core components.

**opsi web service extension**

The heart of the opsi-Nagios-Connector are extended features of the opsi web service. These web service extension make it possible to run checks via web service on the opsi server. So the Nagios server calls checks via web service which are executed on the opsi-server and the results come back to the Nagios server via opsi web service. The advantage of this solution is that there is nearly nothing to do on the monitored opsi server.

The focus of the opsi web service extension lies on opsi specific checks like e.g. rollout monitoring. For the *normal* server monitoring you should use still standard check methods.

**opsi-client-agent extension**

An other part of the opsi-Nagios-Connector is an extension of the opsi-client-agent. In a opsi environment on every managed client runs a opsi-client-agent. With this extension you may use the opsi-client-agent as Nagios agent as well. But in fact not all features of a standard Nagios agent like **NSClient++** are implemented at the opsi-client-agent. You may use the opsi-client-agent to run command line programs and send back the output.

If you not use all functions like NSCA but rather some standard checks per plugin on the client or a set of own plugins on the clients you can use the opsi-client-agent.

If you need more features for the client monitoring you should rollout a standard agent like **NSClient++** via opsi.

The advantage of using the opsi-client-agent as Nagios agent is, that you don't need an additional agent on the client and that you don't need any access data for the clients at the monitoring server. These data is not needed because all check run via the opsi server. This makes the configuration a lot more easier.

9.13.4. opsi-checks

The following chapter explains the goals and configurations of the opsi-checks.
Some background information about where to run the checks

Monitoring administrators know the difference between active and passive checks.

With the opsi-Nagios-Connector we get a new difference: direct and indirect.

- direct:
  The check which collects information about a client runs on that client, get the information direct from the client and sends the information back.

- indirect:
  The check which collects information about a client runs on the opsi server and get the information from the opsi configuration data which is stored in the opsi backend. So - this information may be different from the actual situation of the client.

A good example for an indirect check is the check-opsi-client-status. This check gives you for a given client information about pending action request and reported failures of the opsi software deployment. So this are information about the client from the opsi servers point of view. Therefore this check runs on the opsi server and is an indirect check. A check which runs on the client is a direct check.

For a correct distribution and configuration of the checks you have to analyze your opsi installation. According to the flexibility of opsi many various opsi configurations are possible. So here we can only explain some typical situations. Of course we will get help for special situations by our commercial support.

only one opsi server:

The opsi stand alone installation is the situation that you will find at the most opsi environments. At this installation the opsi config server functionality is at the same server like the (one and only) opsi depot server functionality.

This means to you, that you may ignore if a check has to be run on the config server or the depot server.

opsi with multiple depot servers:

If you have a central management of a multi location opsi environment (one config server, multiple depot servers) the situation is more complicated. So you have to understand the situation:
As the figure points out there is only one server which have data storage for the configuration data - the data backend. This is the opsi config server. The opsi depot server has no own data storage but a redirected backend. This means that if you ask a depot server for any configuration data, this question will be redirected to the config server. And this leads to the consequence that every check which runs against the opsi data backend will at least run on the config server. So you should address checks that run against the backend always to the config server. Even in the situation if you are collecting information about clients which are assigned to a depot which is different from the config server and the check is logically part of the check of this depot server.

If you running direct checks you normally also address the config server. You may address the depot server if the clients can't be reached by the opsi config server via port 4441. In this case it is a good idea to address the depot server.
opsi-check-plugin

At the nagios server there is only one opsi-check-plugin which provides a wide range of different checks. According to the number of features there is also a big number of command line options. So just list all these options won’t help you much. Instead the option will be explained in the context of documentation of the possible checks.

How ever to get a listing of all options you may call check_opsi with the parameters --help or -h.

The following general options are needed for every check:

Table 25. General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-H, --host</td>
<td>opsi server which should run the check</td>
<td>configserver.domain.local</td>
</tr>
<tr>
<td>-P, --port</td>
<td>opsi webservice port</td>
<td>4447 (Default)</td>
</tr>
<tr>
<td>-u, --username</td>
<td>opsi monitoring user</td>
<td>monitoring</td>
</tr>
<tr>
<td>-p, --password</td>
<td>opsi monitoring password</td>
<td>monitoring123</td>
</tr>
<tr>
<td>-t, --task</td>
<td>opsi check method (case sensitive)</td>
<td></td>
</tr>
</tbody>
</table>

The following chapter describes how to call the opsi-check-plugin is called on the command line. How you have to configure these calls at your Nagios server is described at the chapter configuration.

In order to install the opsi-check-plugin on your Nagios server you should add the opsi repository to your server and install the package opsi-nagios-plugins.

For example at Debian or Ubuntu with the following commands:

```
apt-get install opsi-nagios-plugins
```

On RedHat/Centos Servers please use the follwing command:

```
yum install opsi-nagios-plugins
```

And last but not least for openSUSE/SLES Installations:

```
yypper install opsi-nagios-plugins
```

The plugin it self is written in python and should ran at any distribution.

The package bases on the package nagios-plugins-basic respectivly nagios-plugins and installs the plugin to /usr/lib/nagios/plugins.

According to the flexibility of the check_plugin there is no automatic configuration.
Check: opsi web service

This check monitors the opsi web service process (opsiconfd). This check returns also performance data. You should run this check on every opsi server because every opsi server have a opsiconfd process.

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiWebservice
```

This check return normally OK. You will get other return values in the following situations:

- **Critical:**
  - If the opsiconfd is in trouble and can’t reply correctly.
  - If the opsiconfd consumes more than 80% of the cpu.
  - If you have a rate of RPC errors of more than 20%.

- **Warning:**
  - If the opsiconfd consumes more than 60% (but less than 80%) of the cpu.
  - If you have a rate of RPC errors of more than 10% but less than 20%

- **Unknown:**
  The opsi web service could not be reached.

NOTICE: The percentage value of the cpu consumption belongs always to one cpu because the opsiconfd only may use one cpu. (This may change with the opsi multi processing extension)

Check: opsi web service pnp4nagios template

For the display of performance data there is a template for pnp4nagios which displays the data in a combined way. Here is not described how to install pnp4nagios. We assume that pnp4nagios is installed and configured correctly. The way you have to use to configure our template may differ from the below described way according to your pnp4nagios installation (which may use different path).

Standard templates display for every performance data an own diagram. To create a combined display you have to go the following steps:

Step 1:
create at `/etc/pnp4nagios/check_commands` a file named `check_opsiwebservice.cfg` and insert the following content:

```
CUSTOM_TEMPLATE = 0
DATATYPE = ABSOLUTE,ABSOLUTE,ABSOLUTE,ABSOLUTE,DERIVE,GAUGE,GAUGE,GAUGE
```
Step 2:
change to the directory `/usr/share/pnp4nagios/html/templates` and place there a file `check_opsiwebservice.php` which you check out from svn.opsi.org:

```
cd /usr/share/pnp4nagios/html/templates
svn co https://svn.opsi.org/opsi-pnp4nagios-template/trunk/check_opsiwebservice.php
```

Please check that your php file is named exactly like the `command_name` which is defined at the `/etc/nagios3/conf.d/opsi/opsicommands.cfg`. If the names don’t match, a standard template will be used instead our combined template.

After installing this template you should delete the RRD data bases which belong to this check (if there any existing). You will find these data bases at `/var/pnp4nagios/perfdata/<host>/` where you should (only) delete the `opsi-webservice.rrd` and `opsi-webservice.xml` files.

If you have configured everything correctly you should now able to see diagrams like the following screenshot.

![Check: opsi-check-diskusage](image)

**Check: opsi-check-diskusage**

This check monitors the usage of the resources (directories) which are used by opsi. The following table shows the resource names and the corresponding directories:
Table 26. opsi resources

<table>
<thead>
<tr>
<th>Resource name</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>/usr/share/opsiconfd/static</td>
</tr>
<tr>
<td>configed</td>
<td>/usr/lib/configed</td>
</tr>
<tr>
<td>depot</td>
<td>/var/lib/opsi/depot</td>
</tr>
<tr>
<td>repository</td>
<td>/var/lib/opsi/repository</td>
</tr>
</tbody>
</table>

Please note that this check monitors only opsi relevant data and do replace a general disk usage check for the server.

The following command retrieves all resources at one time:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDiskUsage
```

In addition to this standard variant you may restrict the check to the resource `repository`:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDiskUsage --resource repository
```

The default result value of this check is `OK` and the free space of the resources. The free space is given in Gigabyte. The default values for the `Warning` and `Critical` results are:

- **WARNING**: If at least one resource have 5GB or less free space.
- **CRITICAL**: If at least one resource have 1GB or less free space.

This are the default thresholds. They may changed by giving other values for `Warning` with the `-W` or `--warning` options and for `Critical` wit the `-C` or `--critical` option. With these options you can give the thresholds as Gigabyte (G) and as percent (%) as well. The produced output uses the same unit which is used to define the thresholds.

Finally an example:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDiskUsage --resource repository --warning 10% --critical 5%
```

Check: opsi-client-status

One of the targets of the opsi Nagios connector is the software roll out monitoring by viewing to single clients. This is one of the checks which is designed for this job. More exactly: the `software roll out` and `last seen` situation of a certain client is checked.

The result of the following checks is determined by two different states:
• The roll out state of one or more software products:
  The software roll out state results to:
  ◦ **OK** if the software is installed at the in the same product and package version which is available at the server and no action request is set.
  ◦ **Warning** if the software is installed in version that is different to the servers version or if any action request is set.
  ◦ **Critical** if there is a failed reported by the last action.

• The time since last seen:
  The time since last seen results to:
  ◦ **OK** if the client has been seen less or equal then 30 days before.
  ◦ **Warning** if the client has been seen more then 30 days before.

This check may used in different variants, here is the simplest one, which includes all software packages:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkClientStatus -c opsiclient.domain.local
```

As variant it is possible to exclude products from the check. For example:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkClientStatus -c opsiclient.domain.local -x firefox
```

In the example above the product *firefox* was excluded from the check. So this check would not switch to critical because the last action on *firefox* reported a failure.

**Check: opsi-check-ProductStatus**

An other target of the opsi Nagios connector is the software roll out monitoring by viewing to single product or a group of products.

The result of the following checks is determined by the following states:

The software roll out state results to: * **OK** if the software is installed at the in the same product and package version which is available at the server and no action request is set. * **Warning** if the software is installed in version that is different to the servers version or if any action request is set. * **Critical** if there is a failed reported by the last action.

This checks has many variants and is so very flexible. The best way to explain these variants are examples.

The simplest variant check one product on all clients. Here you have to give the product as the opsi **productId**.
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -e firefox

In a simple one server opsi environment, this check is all you need to check the state of the product *firefox* on every client. You will get the information how many clients are in *Warning* and in *Critical* states.

To get the information which clients exactly have the problems, you should call the check in the **verbose mode:**

check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -e firefox -v

An other variant is, that you may exclude a client from the check.

//// produkt muss angegeben werden ?! \///

check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -e firefox -x client.domain.local

In a opsi environment with multiple depot servers you have to use additional options to check also the clients that are not assigned to the config servers depot. If you have multiple depots, you may give the depots name as parameter:

check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -e firefox -d depotserver.domain.local

The reason is that the version of the software packages may differ between your depots. So every client has to be checked against the versions at the depot where they are assigned to. An advantage is that can place the display of the results to the depot server.

You may give instead of the depot servers name the keyword *all* which means all known depot servers. But this normally make only sense if you have only one or two depots. You may also give a comma separated list of depot servers.

An other way to define the checks is to give the name of a opsi groups. So you may check the software roll out state of all products in a given opsi product group. If you have for example a product group *accounting* you may use the following call:

check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -g accounting

Now you will check all products that are Members of the opsi product group *accounting* by this single
check. Important is to see, that the resolution of the opsi group is done while the check at the opsi server. So you may change the opsi group at the opsi Management interface and so you will change the products that will checked without any changes at the Nagios server.

- Sub groups (groups in groups) will not be resolved.

In the same way it is possible to define the clients that should be checked by giving the name of a opsi client group.

An example for a client group `productiveclients`:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -g accounting -G productiveclients
```

This would check all products of the product group `accounting` at all clients of the client group `productiveclients`.

- Sub groups (groups in groups) will not be resolved.

You may also give a comma separated list of opsi groups.

Finally opsi-Clients can be excluded:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -g buchhaltung -G produktivclients -x client.domain.local
```

**Check: opsi-check-depotsync**

If you are using multiple opsi depots the monitoring of synchronicity is important. Even if your depots are for good reasons not completely synchronize they should be synchrony as much as possible to avoid problems by moving a client from one depot to another.

This check monitors if your depots are synchronize according to product ids, product versions and package versions.

This check returns:

- **OK**
  - If all is in sync.
- **Warning**
  - If there is any difference

You should run this check always on the config server because all the data come from the backend of the config server.
Here are some examples.

The base variant:

```bash
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus
```

This base variant is equivalent to the following call:

```bash
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d all
```

So if you don't give the depots which have to be checked, all known depots will be checked. If you have a lot of depots the interpretation of the result is complicated, so it is a good idea to define a lot of single checks where the depots are given as comma separated list:

```bash
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d configserver.domain.local,depotserver.domain.local
```

With this check you compare all products, that are installed on both depots. Any product which is installed only on one of the depot is ignored and will not effect the result.

If you want to include products which are not installed on all checked depots, you have to use the strictmode switch:

```bash
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d configserver.domain.local,depotserver.domain.local --strictmode
```

Now also differences about missing products will be seen.

If you like to exclude a product from the check (perhaps because this product should be in different versions on different depots) you may do this by using the -x option. Here you may also use a comma separated list:

```bash
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d configserver.domain.local,depotserver.domain.local --strictmode -x firefox,thunderbird
```

This check will not warn if the products firefox or thunderbird or not in sync.

Instead of excluding products you may give an explicit list of products that has to been checked:
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d configserver.domain.local,depotserver.domain.local --strictmode -e firefox,thunderbird

In this case only firefox and thunderbird will be checked. We recommend to use this check variant with strictmode to see if one of the products is missing.

check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSync

**Check: Locked products on depots**

During the installation of a new opsi package on an opsi server a lock will be set for the product on the depot. Once the installation has been successfully completed the lock will be removed. The duration of an opsi package can sometime require an usual amount of time without failing. If the lock is kept for a long time then this can be an indication of installation problems.

This check searches for existing locks on your depots.

This check returns:

- **OK**
  
  If there aren’t currently any locked products on opsi-servers.

- **Warning**
  
  If there is at least one locked product on an opsi-server.

You should run this check always on the config server because all the data comes from the backend of the config server.

This check requires at least the following versions:

*Table 27. Minimum package versions*

<table>
<thead>
<tr>
<th>package</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-nagios-plugins</td>
<td>&gt;=4.1.1.1</td>
</tr>
<tr>
<td>opsiconfd</td>
<td>&gt;=4.1.1.11</td>
</tr>
</tbody>
</table>

**Basic check execution:**

check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 --task checkProductLocks

This is equivalent to the following call:
If you want to limit the checks to specific depots you have to supply them as a comma separated list. The result can get harder to interpret if the output for multiple servers is mixed and our recommendation is to define a check per depot.

The following example checks for the two depots `configserver.domain.local` and `depotserver.domain.local`:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 --task checkProductLocks --depotIds configserver.domain.local,depotserver.domain.local
```

By default all products will be checked It is possible to only limit the check to specific products. This takes a comma-separated list of product IDs.

This example will check two specific servers and limit the check to the products `opsi-client-agent` and `opsi-winst`:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 --task checkProductLocks --depotIds configserver.domain.local,depotserver.domain.local --productIds opsi-client-agent,opsi-winst
```

**Check: nagios client plugin check via opsiclientd**

This check gives you an easy possibility to integrate checks that collects the data direct on the client with a minimum of configuration work.

So this check tells the opsiclientd which is running at the opsi client to run a command, fetch the output and send it back.

This extension is not intended to be a complete replacement of a full featured Windows Nagios agent. It is only a light weight alternative.

The plugins which the opsiclientd may call must be compatible to the Nagios plug-in development guidelines. (More details at: [http://nagiosplug.sourceforge.net/developer-guidelines.html](http://nagiosplug.sourceforge.net/developer-guidelines.html)).

In order to run such a plugin on the client, it has to be installed at the client. This problem you will solve by deploying it as an opsi package. The path where the plugin is installed at client doesn’t matter because you have to give the complete path at check definition. We recommend to install all plugins in one directory to ease the maintenance of the plugins at the client.

For security reasons you should make sure that non privileged users have no write access to the plugins, because they will be executed from the opsiclientd with system privileges.
There are lot of ready to use plugins at the internet. One important address to look is: http://exchange.nagios.org/

In the following we assume that your plugins are installed at C:\opsi.org\nagiosplugins\ and we will find ther the plugin check_win_disk.exe out of the package nagioscol from http://sourceforge.net/projects/nagiosplugincol/

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkPluginOnClient --plugin "C:\opsi.org\nagiosplugincol\check_win_disk.exe C:" -c client.domain.local
```

This call checks the client client.domain.local. At the client the plugin check_win_disk.exe is called with the parameter C:. This means, that the hard drive with the letter C should be checked. The output and the result value of the plugin will be fetched by the opsiclientd and will be given back to the Nagios server (via the opsi server) in a for Nagios correct format.

Another special feature is to hold the last check results, even if the client is not reachable.

This feature was implemented according to the fact that desktop clients not always are running like servers, but the most time in their life are usually switched off. Normally Nagios will show for switched off clients the result Unknown. In fact the most problems on the monitored clients will not disappear by just switching them off and on again. So the information that a client had a problem before it was switched off may be an essential information for the system administrator. (You may try to solve this problem by using Timeperiods at the Nagios configuration, but we think that this is not flexible enough and leads to a permanent configuration work). So this opsi extension give you the possibility to give back the last real check results if the client is not reachable right now.

In order to use this feature, you have to use the Nagios macros $SERVICESTATEID$ and $SERVICEOUTPUT$. $SERVICESTATEID$ gives the last result value and should be passed to the -s Option. $SERVICEOUTPUT$ gives the last output line and should be passed to the -o Option. So check can give these last values instead of Unknown if the client is not reachable.

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkPluginOnClient --plugin "C:\opsi.org\nagiosplugincol\check_win_disk.exe C:" -c client.domain.local -s $SERVICESTATEID$ -o $SERVICEOUTPUT$
```

9.13.5. opsi monitoring configuration

This chapter focuses on the configuration that have to been made for a working interface between the opsi and the Nagios server. Just see this as a recommendation, there will be a lot of other ways to do the job.

This description uses a Nagios server as monitoring server. On a Icinga server it should work very similar but you have to change some path entries. It should also work on other Nagios derivatives but this is not tested.
The configuration files from these Chapter are in opsi-nagios-connector-utils svn-Repository. To get these example configuration files you can connect over a browser to following url:


or you can make a direct checkout from repository with following command:

svn co https://svn.opsi.org/opsi-nagios-connector-utils

**opsi monitoring user**

In monitoring environments you will often find that the access is just restricted by IP numbers. Because of the lack of security of this solution we decided to work with a real user / password security in this opsi extension.

Using the opsi standard group `opsiadmin` would give the Nagios more rights than needed. So you have to create an own opsi user for the opsi-Nagios-Connector.

In the following example a user named `monitoring` with the password `monitoring123` is created for opsi:

```
opsi-admin -d method user_setCredentials monitoring monitoring123
```

The created user `monitoring` will be stored with its encrypted password at the `/etc/opsi/passwd` and is not a user which may be used to login at a shell. In fact it is no real Unix user.

You have to create this user only on your config server, even if you have multiple depots.

At your Nagios server you should mask the user and password by making an entry at the `/etc/nagios3/resource.cfg`. This should look for example like this:

```
$USER2$=monitoring
$USER3$=monitoring123
```

The number behind `$USER` may vary. If this configuration was not used before, there should be only `$USER1$` be used. According to what you are using here, you might have to change the other examples in this manual.

**opsi-Nagios-Connector configuration directory**

To make the maintenance of the Nagios configuration easier, we recommend to put all `opsi nagios connector` related configuration files in one separated place.
So just create below `/etc/nagios3/conf.d` a new directory named `opsi` for these configurations.

The configuration files we will place in this directory are:

- Nagios Template: `opsitemplates.cfg`
- Hostgroups: `opsihostgroups.cfg`
- Server Hosts: `<full name of the server>.cfg`
- Commands: `opsicommands.cfg`
- Contacts: `opsicontacts.cfg`
- Services: `opsiservices.cfg`

All the client configuration files we recommend to put in sub directory of this place. Therefore you create below `/etc/nagios3/conf.d/opsi` another directory named `clients`.

**Nagios template:** `opsitemplates.cfg`

Using templates is a standard functionality of Nagios which will not explained here. The main advantage is that it makes the single configuration files smaller and easier to read (and write).

Inside of the templates we use some Nagios *custom variables* for often used values. According to the fact, that the most checks have to run on the opsi config server, we will define the name and port of the config server as such a *custom variable*:

```
_configserver           configserver.domain.local
_configserverurl        4447
```

You will find this below in the template definitions.

These *custom variables* may later on be referenced by the Nagios macros: `$HOSTCONFIGSERVER$` and `$HOSTCONFIGSERVERPORT$`. (These macros have leading `HOST` in their name, because they are defined inside of a host definition).

For more details on variable and macro take look at your Nagios documentation.

Now the first file we create in `/etc/nagios3/conf.d/opsi` is the template definition file `opsitemplates.cfg`.

This file may hold different templates. Every template is created according to the following patter (which contains comments for better understanding):
define host{
    name                opsihost-tmp    ; The name of this host template
    notifications_enabled           1       ; Host notifications are enabled
    event_handler_enabled           1       ; Host event handler is enabled
    flap_detection_enabled          1       ; Flap detection is enabled
    failure_prediction_enabled      1       ; Failure prediction is enabled
    process_perf_data               0       ; Process performance data
    retain_status_information       1       ; Retain status information across
    program restarts
    retain_nonstatus_information    1       ; Retain non-status information across
    max_check_attempts              10
    notification_interval           0
    notification_period             24x7
    notification_options            d,u,r
    contact_groups                  admins
    register                        0       ; DONT REGISTER THIS DEFINITION - ITS
    NOT A REAL HOST, JUST A TEMPLATE!
    icon_image                      opsi/opsi-client.png
}

NOTE: * The optional option icon_image may put it to an image with relative path below: /usr/share/nagios3/htdocs/images/logos/. * Optional you may give an own contact_group, which have to be defined as contact object, for example in the file opsicontacts.cfg.

Now we recommend to create templates for the following objects

- opsi server
- opsi client
- opsi service
- and 2 templates for pnp4nagios (host-pnp / srv-pnp)

Let's start with the example of the opsi server template:
define host{
    name                            opsi-server-tmpl
    notifications_enabled           1
    event_handler_enabled           1
    flap_detection_enabled          1
    failure_prediction_enabled      1
    process_perf_data               1
    retain_status_information       1
    retain_nonstatus_information    1
    check_command                   check-host-alive
    max_check_attempts              10
    notification_interval           0
    notification_period             24x7
    notification_options            d,u,r
    contact_groups                  admins,opsiadmins
    _configserver                   configserver.domain.local
    _configserverport               4447
    register                        0
    icon_image                      opsi/opsi-client.png
}

You just have to change `configserver.domain.local` to your config server name. Also you may change the `contact_groups` to your needs.

The next part of the file `opsitemplates.cfg` is the template for the clients:

define host{
    name                            opsi-client-tmpl
    notifications_enabled           1
    event_handler_enabled           1
    flap_detection_enabled          1
    failure_prediction_enabled      1
    process_perf_data               1
    retain_status_information       1
    retain_nonstatus_information    1
    max_check_attempts              10
    notification_interval           0
    notification_period             24x7
    notification_options            d,u,r
    contact_groups                  admins,opsiadmins
    _configserver                   configserver.domain.local
    _configserverport               4447
    register                        0
    icon_image                      opsi/opsi-client.png
}
The Option "check command check-host-alive" should be not set here because the clients are not always running. In effect the clients will be displayed as \textit{pending} instead of \textit{offline}.

You just have to change \texttt{configserver.domain.local} to your config server name. Also you may change the \texttt{contact_groups} to your needs.

The next part of the file \texttt{opsitemplates.cfg} is the template for the opsi-services:

```plaintext
define service{
  name                            opsi-service-tmpl
  active_checks_enabled           1
  passive_checks_enabled          1
  parallelize_check               1
  obsess_over_service             1
  check_freshness                 0
  notifications_enabled           1
  event_handler_enabled           1
  flap_detection_enabled          1
  failure_prediction_enabled      1
  process_perf_data               1
  retain_status_information       1
  retain_nonstatus_information    1
  notification_interval           0
  is_volatile                     0
  check_period                    24x7
  normal_check_interval           5
  retry_check_interval            1
  max_check_attempts              4
  notification_period             24x7
  notification_options            \texttt{w,u,c,r}
  contact_groups                  admins,opsiadmins
  register                        0
}
```

If you are using pnp4nagios for the graphic display of performance data you will need two other templates in the file \texttt{opsitemplates.cfg}:
define host {
    name     host-pnp
    action_url /pnp4nagios/index.php/graph?host=$HOSTNAME$&srv=_HOST_
    register 0
}
define service {
    name     srv-pnp
    action_url /pnp4nagios/index.php/graph?host=$HOSTNAME$&srv=$SERVICEDESC$
    register 0
}

**opsi hostgroup:** opsihostgroups.cfg

The next step is to define the hostgroups. This helps to structure the display of the results as well as the service definitions.

So create a file named `opsihostgroups.cfg` with the following content:

```
define hostgroup {
    hostgroup_name  opsi-clients
    alias           OPSI-Clients
}
define hostgroup {
    hostgroup_name  opsi-server
    alias           OPSI-Server
    members         configserver.domain.local, depotserver.domain.local
}
```

Do not forget to edit the `member` line.

**opsi server:** `<full name of the server>.cfg`

The next step is to create for every opsi server you are running an own configuration file. This file should be named based on the pattern `<full name of the server>.cfg`. For example `configserver.domain.local.cfg`.

(You may also create one file (e.g. `opsihost.cfg` with all server definitions).

The content should look like this:
define host{
  use             opsi-server-tmpl
  host_name       configserver.domain.local
  hostgroups      opsi-server
  alias           opsi Configserver
  address         configserver.domain.local
}

define host{
  use             opsi-server-tmpl
  host_name       depotserver.domain.local
  hostgroups      opsi-server
  alias           opsi Depotserver
  address         depotserver.domain.local
}

Explanation of the entries: * use references to the used template. * hostgroups tells us to which hostgroup this server belongs.

**opsi Clients:** clients/<full name of the client>.cfg

The opsi client configurations should be placed in an own sub directory. They should be defined like this:

define host{
  use             opsi-client-tmpl
  host_name       client.domain.local
  hostgroups      opsi-clients
  alias           opsi client
  address         client.domain.local
  _depotid        depotserver.domain.local
}

This client configuration uses again a custom variable: _depotid. This custom variable may be referenced by the macro $\_HOSTDEPOTID$. The usage is optional. If a client may be not connected by the opsi configuration server directly, you will here write down from which depot server the client can be contacted.

To make it easier to create the configuration files for your large number of opsi clients, you may run the following script on your opsi configuration server:
import python

```
from OPSI.Backend.BackendManager import *

template = '''
define host {
    use             opsi-client-tmpl
    host_name       %hostId%
    hostgroups      opsi-clients
    alias           %hostId%
    address         %hostId%
}
...
```

backend = BackendManager(
    dispatchConfigFile = u'/etc/opsi/backendManager/dispatch.conf',
    backendConfigDir   = u'/etc/opsi/backends',
    extensionConfigDir = u'/etc/opsi/backendManager/extend.d',
)

hosts = backend.host_getObjects(type="OpsiClient")

for host in hosts:
    filename = "%s.cfg" % host.id
    entry = template.replace("%hostId%",host.id)
    f = open(filename, 'w')
    f.write(entry)
    f.close()

**opsi command configuration:** opsicommands.cfg

Now we have to define which of the check commands, which are described before, we want to use. You should do this in a file named `opsicommands.cfg`. This is just an example which you may change to your needs:

First let us explain the structure of the entries:

```
define command{
    command_name    check_opsi_clientstatus
    command_line        $USER1$/check_opsi -H $_HOSTCONFIGSERVER$ -P
                       $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkClientStatus -c $HOSTADDRESS$
}
```

The **command_name** will be used by other configuration files. The option **command_line** defines the command and all used arguments.
Based on this pattern we create now the file \texttt{opsicommands.cfg}:

```
define command {
    command_name    check_opsiwebservice
    command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTADDRESS$ -P 4447 -u $USER2$ -p $USER3$ -t checkOpsiWebService
}
define command {
    command_name    check_opsidiskusage
    command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTADDRESS$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkOpsiDiskUsage
}
define command {
    command_name    check_opsiclientstatus
    command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkClientStatus -c $HOSTADDRESS$
}
define command {
    command_name    check_opsiproductstatus
    command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkProductStatus -e $ARG1$ -d $HOSTADDRESS$ -v
}
define command {
    command_name    check_opsiproductStatus_withGroups
    command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkProductStatus -g $ARG1$ -G $ARG2$ -d "all"
}
define command {
    command_name    check_opsiproductStatus_withGroups_long
    command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkProductStatus -g $ARG1$ -G $ARG2$ -v -d "all"
}
define command {
    command_name    check_opsidepotsync
    command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkDepotSyncStatus -d $ARG1$
}
define command {
    command_name    check_opsidepotsync_long
    command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkDepotSyncStatus -d $ARG1$ -v
}
define command {
    command_name    check_opsidepotsync_strict
    command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkDepotSyncStatus -d $ARG1$ -v
}
```
Contacts: opsicontacts.cfg

This define the contacts which will get notifications.

```plaintext
$HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkDepotSyncStatus -d $ARG1$ --strict
}
define command {
    command_name    check_opsidepotsync_strict_long
    command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTCONFIGSERVER$ -P $HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkDepotSyncStatus -d $ARG1$ --strict -v
}
define command {
    command_name    check_opsipluginon_client
    command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTCONFIGSERVER$ -P $HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkPluginOnClient -c $HOSTADDRESS$ --plugin $ARG1$
}
define command {
    command_name    check_opsipluginon_client_with_states
    command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTCONFIGSERVER$ -P $HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkPluginOnClient -c $HOSTADDRESS$ --plugin $ARG1$ -s $SERVICESTATEID$ -o "$SERVICEOUTPUT$"
}
define command {
    command_name    check_opsipluginon_client_from_depot
    command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTDEPOTID$ -P $HOSTCONFIGSERVERPORT$ -u $USER2$ -p $USER3$ -t checkPluginOnClient -c $HOSTADDRESS$ --plugin $ARG1$
}
define contact{
    contact_name                    adminuser
    alias                           Opsi
    service_notification_period     24x7
    host_notification_period        24x7
    service_notification_options    w,u,c,r
    host_notification_options       d,r
    service_notification_commands   notify-service-by-email
    host_notification_commands      notify-host-by-email
    email                           root@localhost
}
define contactgroup{
    contactgroup_name       opsiadmins
    alias                   Opsi Administrators
    members                 adminuser
}
```
You should replace *adminuser* by one or more real users.

**Services:** *opsiservices.cfg*

Finally we define with the *services* what the Nagios server have to monitor and to display. This definition are using the definition of the other configuration file above like templates, commands and hostgroups or hosts.

As first part we define the services which give us information’s about the servers. One of these is the check if the depots are in sync, which is here down against *all* known depots.

```plaintext
#OPSI-Services
define service{
    use                             opsi-service-tmpl,srv-pnp
    hostgroup_name                  opsi-server
    service_description             opsi-webservice
    check_command                   check_opsiwebservice
    check_interval                  1
}
define service{
    use                             opsi-service-tmpl
    hostgroup_name                  opsi-server
    service_description             opsi-diskusage
    check_command                   check_opsidiskusage
    check_interval                  1
}
define service{
    use                             opsi-service-tmpl
    hostgroup_name                  opsi-server
    service_description             opsi-depotsyncstatus-longoutput
    check_command                   check_opsidepotsync_long!all
    check_interval                  10
}
define service{
    use                             opsi-service-tmpl
    hostgroup_name                  opsi-server
    service_description             opsi-depotsyncstatus-strict-longoutput
    check_command                   check_opsidepotsync_strict_long!all
    check_interval                  10
}
```

The next part is the monitoring of the software roll out. In one check a concrete opsi product *opsi-client-agent* is mentioned. In two other check are referenced on a opsi product group *opsiessentials* and opsi client group *productiveclients*. 
define service{
    use opsi-service-tmpl
    hostgroup_name opsi-clients
    service_description opsi-clientstatus
    check_command check_opsiclientstatus
    check_interval 10
}

define service{
    use opsi-service-tmpl
    hostgroup_name opsi-server
    service_description opsi-productstatus-opsiclientagent
    check_command check_opsiproductstatus!opsi-client-agent
    check_interval 10
}

define service{
    use opsi-service-tmpl
    hostgroup_name opsi-server
    service_description opsi-productstatus-opsiessentials-group
    check_command check_opsiproductStatus_withGroups!opsiessentials!productiveclients
    check_interval 10
}

define service{
    use opsi-service-tmpl
    hostgroup_name opsi-server
    service_description opsi-productstatus-opsiessentials-group-longoutput
    longoutput
    check_command check_opsiproductStatus_withGroups_long!opsiessentials!productiveclients
    check_interval 10
}

In the third and last part of the file, the checks which are should run directly on the clients (direct checks) are defined.
These checks are (for example) not assigned to hostgroups but to single hosts or lists of hosts (client.domain.local, depotclient.domain.local).

Some description:

- **opsi-direct-checkpluginonclient**
  runs a normal direct check on the client and results to unknown if the client is offline.
  At this check the config server try's to reach the client directly.
- **opsi-direct-checkpluginonclient-with-servicestate**
  is equal to opsi-direct-checkpluginonclient, but returns the last valid result if the client is offline (instead of unknown)
- **opsi-direct-checkpluginonclient-from-depot**
is equal to `opsi-direct-checkpluginonclient`, but the client will be connected by the server which is given in the host configuration as `_depotid`.

```plaintext
define service{
    use                     opsi-service-tmpl
    host_name               client.domain.local,depotclient.domain.local
    service_description     opsi-direct-checkpluginonclient
    check_command
        check_opsipluginon_client!"C:\\opsi.org\\nagiosplugins\\check_memory.exe"
        check_interval   10
}
define service{
    use                     opsi-service-tmpl
    host_name               client.domain.local
    service_description     opsi-direct-checkpluginonclient-with-servicestate
    check_command
        check_opsipluginon_client_with_states!"C:\\opsi.org\\nagiosplugins\\check_memory.exe"
        check_interval   10
}
define service{
    use                     opsi-service-tmpl
    host_name               depotclient.domain.local
    service_description     opsi-direct-checkpluginonclient-from-depot
    check_command
        check_opsipluginon_client_from_depot!"C:\\opsi.org\\nagiosplugins\\check_memory.exe"
        check_interval   10
}
```

### 9.14. opsi-clonezilla (free)

#### 9.14.1. Preconditions for the opsi Extensions opsi-clonezilla

Technical preconditions are opsi 4.0.3 with the following package and product versions:

<table>
<thead>
<tr>
<th>opsi-Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-linux-bootimage</td>
<td>&gt;= 20130207-1</td>
</tr>
</tbody>
</table>

or opsi 4.0.5 with the following package and product versions:

<table>
<thead>
<tr>
<th>opsi packet</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-linux-bootimage</td>
<td>&gt;= 20140805-1</td>
</tr>
</tbody>
</table>
For the product `opsi-clonezilla` the share `opsi_images` must have write permission for `pcpatch`. Check your Samba configuration. For use with UEFI you need at least `opsi 4.0.7`.

Set for the property `imageshare` a share as value. This share should have the format `//server/share`. Please note the use of slashes instead of back slashes. This share should be mountable by the opsi user `pcpatch` with the password as known by the opsi-server. This is normally the share `opsi_images` from the opsi-server.

### 9.14.2. Introduction

Besides of the package based (unattended) installation, opsi had in the past just a rudimentary support for image based installations. With the integration technique of the Open Source product `clonezilla` ([http://clonezilla.org/](http://clonezilla.org/)) into opsi, now a comprehensive and flexible solution for handling partition and disc images is available.

### 9.14.3. Concept

We have combined the `clonezilla` scripts with the opsi-linux-bootimage to generate the following benefits:

- integration into the opsi process control
- automated mount of the shares for the image repository
- availability of automated processing

### 9.14.4. Interactive Proceedings

Starting the `opsi-clonezilla` per default starts in the interactive mode. This interactive mode allows to choose the desired operations and parameters easily. Knowing the commands and their parameters from this makes it easy to create non-interactive run commands from this.

- Set for the property `imageshare` a share as value. This share should have the format `//server/share`. Please note the use of slashes instead of back slashes. This share should be mountable by the opsi user `pcpatch` with the password as known by the opsi-server. This is normally the share `opsi_images` from the opsi-server.

- Switch the property `runcommand` to `ocs-live`. This is the interactive mode of `clonezilla`.

- Start the netboot product.

- In the first dialog you will be asked, whether anything should be mounted to `/home/partimg`. Choose `Skip` because the mount has already been done by the opsi bootimage.
Figure 121. Skip: The share given by the property imageshare will be mounted by the bootimage to /home/partimg.

The mounted partition will be displayed:

By choosing Expert or Beginner you decide if you want to use all default parameters or have the possibility to modify them.

Since opsi 4.0.5 you may also choose the Beginner mode.
Now you have to choose which basic operation you like to run. In this manual we discuss only the following operations:

- save disk
- save partition
- restore disk
- restore partition

**Figure 124. Choose operation.**

**Interactive save disk in the expert mode**

Here will be shown (as an example for similar operations) the additional dialogs you will get in the save disk expert mode.

**Figure 125. Choose the tools (default value recommended)**

**Figure 126. miscellaneous: unset -c here to suppress interactive questions for automation.**
Figure 127. Compression method before opsi 4.0.5, which is bootimages \( \approx 20130207 \) (opsi-clonezilla_2.01-3), select here -z1. With opsi 4.0.5 and above is not required anymore.

Figure 128. Check filesystem (den default skip nutzen)

Figure 129. Check the saved image (den default yes nutzen)

Figure 130. Action after cloning (use the default -p true, the reboot is triggered by the opsi bootimage).

Interactive save disk
Figure 131. Name for the image to be saved on disc.

Choose local disk as source.
The disk name is the device name in GNU/Linux. The first disk in the system is "sda" or "sdn", the 2nd disk is "sdb" or "sdd".

Figure 132. Select the disc to create the image from.

The resulting command. This can be set as product property runcommand.
9. opsi Extensions

Reading the partition table for /dev/sda...RTUAL-0
The first partition of disk /dev/sda starts at 2048.
Saving the hidden data between MBR (1st sector, i.e. 512 bytes) and 1st partition, which may by:
```
dd if=/dev/sda of=/home/partimag/partimg/sda-hidden-data-after-mbr skip=1 bs=512 count=2047
```
2047=0 records in
2047=0 records out
1048064 bytes (1.0 MB) copied, 0.0257349 s, 40.7 MB/s

Saving the MBR data for sda...
140 records in
140 records out
512 bytes (512 B) copied, 0.00170123 s, 301 kB/s

Starting saving /dev/sda1 as /home/partimg/partimg/sda1.XXX.../dev/sda1 filesystem: ntfs.

Checking the disk space...
Use partclone with gzip to save the image.
Image file will be split with size limit 2000 MB.

If this action fails or hangs, check:
* Is the disk full?

Partclone v0.2.8 http://partclone.org
Starting to clone device (/dev/sda1) to Image (-)
Reading Super Block
Elapsed: 00:00:04, Remaining: 00:00:00, Completed: 100.00%,
Total Time: 00:00:04, 100.00% completed!
File system: NTFS
Device size: 21.5 GB
Space in use: 13.8 GB
Free Space: 7.7 GB
Block size: 4096 Byte
Used block: 3371140
Elapsed: 00:00:35, Remaining: 00:08:55, Completed: 6.14%, Rate: 1.456GB/min.

Figure 134. Progress bar

Interactive save part

![Clonzilla - Opensource Clone System (OCS) | Mode: savedisk]

Input a name for the saved image to use

2014-06-11-12-img

<Ok>  <Cancel>

Figure 135. Name for the partition image to be saved as.
9. opsi Extensions

Figure 136. Select the partition to create the image from

```bash
[drsl-1]# partimage sdal

Figure 137. The resulting command. This can be set as product property runcommand

```bash
Reading the partition table for /dev/sda...RETURN=0

The first partition of disk /dev/sda starts at 2048.

Sanding the hidden data between MBR (1st sector), i.e. 512 bytes) and 1st partition, which will be
by:
4d if /dev/sda of=/home/partimag/partimg/sdal-hidden-data-after-mbr skip=1 bs=512 count=2047
2047+0 records in
2047+0 records out
1048576 bytes (1.0 MB) copied, 0.0257349 s, 40.7 MB/s

Saving the MBR data for sda...
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.00176123 s, 301 kB/s

Starting saving /dev/sdal as /home/partimag/partimg/sdal.XXX.../
/dev/sdal filesystem: ntfs.

Checking the disk space...
Use partclone with gzip to save the image.
Image file will be split with size limit 2000 MB.

If this action fails or hangs, check:
* Is the disk full?

Partclone v6.2.8 http://partclone.org
Starting to clone device /dev/sdal to image (-)
Reading Super Block
Calculating bitmap...
Elapsed: 00:00:01, Remaining: 00:00:00, Completed: 100.00%,
Total Time: 00:00:01, 100.00% completed!
File system: ntfs
Device size: 21.5 GB
Space in use: 13.8 GB
Free space: 7.7 GB
Block size: 4096 Byte
Used block: 3393440
Elapsed: 00:00:35, Remaining: 00:00:55, Completed: 6.14%, Rate: 1.45GB/min.

Figure 138. Progress bar

Interactive restore disk
9. opsi Extensions

Figure 139. Select the disc image to be restored

```plaintext
Choose the image file to restore:

2014-06-11-12-img 2014-06-11-1831 sda
<Ok> <Cancel>
```

Figure 140. Select the disc where the image is to be restored

```plaintext
Choose the target disk(s) to be overwritten (ALL DATA ON THE ENTIRE DISK WILL BE LOST AND REPLACED!!)
The disk name is the device name in GNU/Linux. The first disk in the system is "hda" or "sda", the 2nd disk is "hdb" or "sdb"... Press space key to mark your selection. An asterisk (*) will be shown when the selection is done:

sda 21.5GB_VBOX_HDD_DISK ata-VBOX_HDD_DISK VBRcf6208-1046a61
<Ok> <Cancel>
```

Figure 141. The resulting command. This can be set as product property runcommand

```plaintext
9. opsi Extensions

Active the partition info in /proc... done!
Setting /dev/sdinfo...
++++++++++++++++++++++++++++++++++++++++++++++++++++++
The following step is to restore an image to the hard disk/partition(s) on this machine: /home/partimg/2014-06-11-12-img -> sda sda1
WARNING!!! WARNING!!! WARNING!!!
WARNING!!! THE EXISTING DATA IN THIS HARD/DISK/PARTITION(S) WILL BE OVERWRITTEN. ALL EXISTING DATA WILL BE LOST.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++
Machine: VirtualBox
sda (21.5GB VBOX_HDD_DISK_ata-VBOX_HDD_DISK VBRcf6208-1046a61)
sda (21.5GB ata_VBOX_HDD_DISK VBRcf6208-1046a61)
+++++++++++++++++++++++++++++++++++++++++++++++++++++++
Are you sure you want to continue? [y/n]
```

Figure 142. Query before starting to overwrite the disc. Can be suppressed by omitting the option -c from the command.
9. opsi Extensions

**Partclone**

Partclone v0.2.0 http://partclone.org
Starting to restore image (-) to device (/dev/sda1)
Calculating bitmap... Please wait... done!

File system: NTFS
Device size: 21.5 GB
Space in use: 12.7 GB
Free Space: 0.7 GB
Block size: 4096 Byte
Used block: 3168766

Elapsed: 00:00:41
Remaining: 00:05:52
Rate: 1.94GB/min

10%

![Figure 143. Progress bar](image)

**Interactive restore part**

Choose the image file to restore:

- 2013-02-27-19-ing 2013-0227-1031_sda1
- 2014-06-11-12-ing 2014-0611-1031_sda1
- partition 2014-0611-1053_sda1
- xpclient-ing 2013-0301-1646_sda1

![Figure 144. Select the part image to be restored](image)

Choose the target partition(s) to be overwritten (ALL DATA ON THE PARTITION WILL BE LOST AND REPLACED!!)
The partition name in the device name is UUID/Linux. The first partition in the first disk is "hda1" or "sda1", the 2nd partition in the first disk is "hda2" or "sda2", the first partitions in the second disk is "hda1" or "sda1"... If the system you want to save is MS windows, normally C: is hda1 (or PS) or sda1 (for PSX), and D: could be hda2 (or sda2), hda5 (or sda5)...

![Figure 145. Select the partition where the image is to be restored.](image)
Figure 146. Query before starting to overwrite the disc. Can be suppressed by omitting the option -c from the command.

Figure 147. Progress bar


By setting the desired command as the product property runcommand opsi-clonezilla is switched to the non interactive mode.

- Set the property imageshare to a share, that can be mounted by the user pcpatch with the password as known by the opsi-server. The format for the share is //server/share (attention: use forward slashes, not backward slashes).

- Set the property runcommand to the non interactive command. Recommended Parameters:
  - Always: --batch
  - At restore: --skip-check-restorable-r
  - Always remove: -c
Here are some non interactive versions of the examples from above (without -c and with --batch). Since opsi 4.0.5 the parameter -z1 can be omitted. This accelerates the compression with multi processor kernels:

- `/opt/drbl/sbin/ocs-sr --batch -q2 -j2 -rm-win-swap-hib -z1 -i 2000 -p true save parts partimg sda1`
- `/opt/drbl/sbin/ocs-sr --batch -g auto -e1 auto -e2 -r -j2 -p true restore disk 2014-06-11-12-img sda`
- `/opt/drbl/sbin/ocs-sr --batch -g auto -e1 auto -e2 -r -j2 -k -p true restore parts partimg sda1`

Furthermore in these examples the image names `2014-06-11-12-img` or `partimg` can be replaced by the string `imagefile`. In this case the string `imagefile` will be substituted by the value of the property `imagefile`.

If you would take the device names `sda` or rather `sda1` for example, and replace them with, the string `diskdevice` or `partdevice`, then the string `disk_number` or `part_number` will be also respectively replaced.

Examples for `disk_number`=1 and `part_number`=1:
- `sda / sda1`
- `cciss/c0d0 / cciss/c0d0p1`

As a result you can look at the following examples:

- `/cs-sr -g auto -e1 auto -e2 --skip-check-restorable-r --batch -r -j2 -p true restoredisk imagefile diskdevice`
- `/cs-sr -q2 --batch -j2 -rm-win-swap-hib -i 2000 -p true savedisk imagefile diskdevice`
- `/cs-sr -q2 -c -j2 -z1 -i 2000 -sc -p true saveparts imagefile partdevice`

9.14.6. opsi-clonezilla properties

- `askbeforeinst`
  - description: Should there be a confirmation dialog before start installing? / Faut-il y avoir une confirmation avant de démarrer l’installation?
  - default: False
- `mount_image_share`
  - description: Should there be a confirmation dialog before start installing? / Faut-il y avoir une confirmation avant de démarrer l’installation?
  - default: True
- `imageshare`
  - editable: True
- description: normally **auto** or empty: Defaults to the opsi_images share of the depot server; if not **auto** or empty: smb/cifs share in the format //server/share
  - values: ["", "/opsiserver/opsi_images", "auto"]
  - default: ["auto"]

**runcommand**
- editable: True
- description: Clonezilla command to be executed
  - default: ["ocs-live"]

**disk_number**
- editable: True
- description: Number (first=1) of the disk; if string **diskdevice** in the runcommand it will be replaced by valid device path (eg sda)
  - values: ["1", "2"]
  - default: ["1"]

**part_number**
- editable: True
- description: Number (first=1) of the partition of **disk_number**; if string **partdevice** in the runcommand it will be replaced by valid device path (eg sda1)
  - values: ["1", "2", "3", "4", "5"]
  - default: ["1"]

**imagefile**
- editable: True
- description: name of the imagefile; will replace the string **imagefile** in the runcommand
  - values: ["myimagefile"]
  - default: ["myimagefile"]

**drbl_ocs_conf**
- editable: True
- description: Directory for post run scripts (Entries in /etc/drbl/drbl-ocs.conf)
  - values: ["", "OCS_POSTRUN_DIR="/home/partimag/postrun"", "OCS_PRERUN_DIR="/home/partimag/prerun"" ]

**rebootflag**
• editable: False
  • description: Should the Client reboot after running the script
  • values: ["keepalive", "reboot", "shutdown"]
  • default: ["reboot"]

• setup_after_install
  • multivalue: True
  • editable: True
  • description: Which opsi product(s) should we switch to setup after clonezilla work is finished?
  • values: ["""]
  • default: [""]

• architecture
  • editable: False
  • description: Selection of architecture, influences the selection of the installation and the installation architecture.
  • values: ["32bit", "64bit"]
  • default: ["32bit"]

9.14.7. opsi-clonezilla known bugs

None


Save and restore of images

http://clonezilla.org/clonezilla-live-doc.php

Clonezilla ocs-sr options

/usr/sbin/ocs-sr:
Usage:
   To save or restore image
ocs-sr [OPTION] {savedisk|saveparts|restoredisk|restoreparts} IMAGE_NAME DEVICE

Options for saving:

-enc, --enc-ocs-img
   To encrypt the image with passphrase.
-fsck-src-part, --fsck-src-part
   Run fsck interactively on the source file system before saving it.

-fsck-src-part-y, --fsck-src-part-y
   Run fsck automatically on the source file system before saving it. This option will always attempt
to fix any detected filesystem corruption automatically. //NOTE:// Use this option in caution.

-gm, --gen-md5sum
   Generate the MD5 checksum for the image. Later you can use -cm|--check-md5sum option to check
the image when restoring the image. Note! It might take a lot of time to generate if the image size is
large.

.gs, --gen-sha1sum
   Generate the SHA1 checksum for the image. Later you can use -cs|--check-sha1sum option to check
the image when restoring the image. Note! It might take a lot of time to generate if the image size is
large.

-gmf, --gen-chksum-for-files-in-dev
   Generate the checksum for files in the source device. Later you can use -cmf|--chk-chksum-for-files
-in-dev to check the files in the destination device after they are restored. Note! It might take a lot
of time to inspect the checksum if there are many files in the destination device.

-i, --image-size SIZE
   Set the size in MB to split the partition image file into multiple volumes files. For the FAT32 image
repository, the SIZE should not be larger than 4096.

-j2, --clone-hidden-data
   Use dd to clone the image of the data between MBR (1st sector, i.e. 512 bytes) and 1st partition,
which might be useful for some recovery tool.

-ntfs-ok, --ntfs-ok
   Assume the NTFS integrity is OK, do NOT check again (for ntfsclone only)

-rm-win-swap-hib, --rm-win-swap-hib
   Try to remove the MS windows swap file in the source partition.

-q, --use-ntfsclone
   If the partition to be saved is NTFS, use program ntfsclone instead of partimage (i.e. Priority:
ntfsclone > partimage > dd)

-q1, --force-to-use-dd
   Force to use dd to save partition(s) (inefficient method, very slow, but works for all the file system).

-q2, --use-partclone
   Use partclone to save partition(s) (i.e. partclone > partimage > dd).
-rescue, --rescue
  Turn on rescue mode, i.e. try to skip bad sectors.

-sc, --skip-check-restorable
  By default Clonezilla will check the image if restorable after it is created. This option allows you to skip that.

-no-compress
  Don't compress when saving: very fast but very big image file (NOT compatible with multicast restoring!!)

-gzip-compress
  Compress using gzip when saving: fast and small image file (default)

-smp-gzip-compress
  Compress using parallel gzip program (pigz) when saving: fast and small image file, good for multi-core or multi-CPU machine

-bzip2-compress
  Compress using bzip2 when saving: slow but smallest image file

-smp-bzip2-compress
  Compress using parallel bzip2 program (lbzip2) when saving: faster and smallest image file, good for multi-core or multi-CPU machine

-lzop-compress
  Compress using lzop when saving: similar to the size by gzip, but faster than gzip.

-lzma-compress
  Compress using lzma when saving: slow but smallest image file, faster decompression than bzip2.

-xz-compress
  Compress using xz when saving: slow but smallest image file, faster decompression than bzip2.

-smp-xz-compress
  Compress using parallel xz when saving: slow but smallest image file, faster decompression than bzip2.

-lzop-compress
  Compress using lzop when saving: slow but smallest image file, faster decompression than bzip2.

-smp-lzop-compress
  Compress using parallel lzop when saving: slow but smallest image file, faster decompression than bzip2.
-z7, --lrzip-compress
Compress using lrzip when saving.

-i, --image-size SIZE
Set the split image file volume size SIZE (MB). When ocs-sr is run with -x, the default SIZE is set as 4096, if without -x, we will not split it. Some words are reserved for IMAGE_NAME, "ask_user" is used to let user to input a name when saving an image. "autoname" is used to automatically generate the image name based on network card MAC address and time. "autohostname" is used to automatically generate the image name based on hostname. "autoproductname" is used to automatically generate the image name based on hardware product model gotten from dmidecode. A word is reserved for DEVICE, "ask_user" could be used to let user to select the source device when saving an image.

Options for restoring:

-f, --from-part-in-img PARTITION
Restore the partition from image. This is especially for "restoreparts" to restore the image of partition (only works for one) to different partition, e.g. sda1 of image to sdb6.

-g, --grub-install GRUB_PARTITION
Install grub in the MBR of the disk containing partition GRUB_PARTITION with root grub directory in the same GRUB_PARTITION when restoration finishes, GRUB_PARTITION can be one of "/dev/hda1", "/dev/hda2"... or "auto" ("auto" will let clonezilla detect the grub root partition automatically). If "auto" is assigned, it will work if grub partition and root partition are not in the same partition.

-r, --resize-partition
Resize the partition when restoration finishes, this will resize the file system size to fit the partition size. It is normally used when when a small partition image is restored to a larger partition.

-k, --no-fdisk, --no-create-partition
Do NOT create partition in target harddisk. If this option is set, you must make sure there is an existing partition table in the current restored harddisk. Default is to create the partition table.

-icrc, --icrc
Skip Partclone CRC checking.

-irhr, --irhr
Skip removing the Linux udev hardware records on the restored GNU/Linux.

-irvd, --irvd
Skip removing the NTFS volume dirty flag after the file system is restored.

-ius, --ius
Skip updating syslinux-related files on the restored GNU/Linux.
-icds, --ignore-chk-dsk-size-pt
Skip checking destination disk size before creating the partition table on it. By default it will be checked and if the size is smaller than the source disk, quit.

iefi, --ignore-update-efi-nvram
Skip updating boot entries in EFI NVRAM after restoring.

-k1,
Create partition table in the target disk proportionally.

-k2,
Enter command line prompt to create partition table manually before restoring image.

-scr, --skip-check-restorable-r
By default Clonezilla will check the image if restorable before restoring. This option allows you to skip that.

-t, --no-restore-mbr
Do NOT restore the MBR (Mater Boot Record) when restoring image. If this option is set, you must make sure there is an existing MBR in the current restored harddisk. Default is Yes

-u, --select-img-in-client
Input the image name in clients

-e, --load-geometry
Force to use the saved CHS (cylinders, heads, sectors) when using sfdisk

-e1, --change-geometry NTFS-BOOT-PARTITION
Force to change the CHS (cylinders, heads, sectors) value of NTFS boot partitoin after image is restored. NTFS-BOOT-PARTITION can be one of "/dev/hda1", "/dev/hda2"... or "auto" ("auto" will let clonezilla detect the NTFS boot partition automatically)

-e2, --load-geometry-from-edd
Force to use the CHS (cylinders, heads, sectors) from EDD (Enhanced Disk Device) when creating partition table by sfdisk

-y, -y0, --always-restore, --always-restore-default-local
Let Clonezilla server as restore server, i.e. client will always has restore mode to choose (However default mode in PXE menu is local boot)

-y1, --always-restore-default-clone
Let Clonezilla server as restore server, i.e. client will always has restore mode to choose (The default mode in PXE menu is clone, so if client boots, it will enter clone always, i.e. clone forever)

-j, --create-part-by-sfdisk
Use sfdisk to create partition table instead of using dd to dump the partition table from saved
-j0, --create-part-by-dd

Use dd to dump the partition table from saved image instead of sfdisk. Note: This does NOT work when logical drives exist.

-j1, --dump-mbr-in-the-end

Use dd to dump the MBR (total 512 bytes, i.e. 446 bytes (executable code area) + 64 bytes (table of primary partitions) + 2 bytes (MBR signature; # 0xAA55) = 512 bytes) after disk image was restored. This is an insurance for some hard drive has different numbers of cylinder, head and sector between image was saved and restored.

-j2, --clone-hidden-data

Use dd to clone the image of the data between MBR (1st sector, i.e. 512 bytes) and 1st partition, which might be useful for some recovery tool.

-hn0 PREFIX

Change the hostname of M$ Windows based on the combination of hostname prefix and IP address, i.e. PREFIX-IP

-hn1 PREFIX

Change the hostname of M$ Windows based on the combination of hostname prefix and NIC MAC address, i.e. PREFIX-MAC

--max-time-to-wait TIME

When not enough clients have connected (but at least one), start anyways when TIME seconds since first client connection have pased. This option is used with --clients-to-wait

-cm, --check-md5sum

Check the MD5 checksum for the image. To use this option, you must enable -gm|--gen-md5sum option when the image is saved. Note! It might take a lot of time to check if the image size is large.

-cs, --check-sha1sum

Check the SHA1 checksum for the image. To use this option, you must enable -gs|--gen-sha1sum option when the image is saved. Note! It might take a lot of time to check if the image size is large.

-cmf, --chk-chksum-for-files-in-dev

Check the checksum for the files in the device. To use this option, you must enable -gmf|--gen-chksum-for-files-in-dev when the image is saved. Note! (1) The file system must be supported by Linux kernel so that it can be mounted as read-only to check the files. (2) It might take a lot of time to check if there are many files in the source device.

-srel, --save-restore-error-log

Save the error log file in the image dir. By default the log file won't be saved when error occurs.
--mcast-port NO
Assign the udp port number for multicast restore. This is used by clonezilla server. Normally it's not necessary to manually assign this option. Some words are reserved for IMAGE_NAME, "ask_user" is used to let user to input a name when saving an image. "autoproductname" is used to automatically get the image name based on hardware product model from dmidecode. A word is reserved for DEVICE, "ask_user" could be used to let user to select the source device when saving an image.

General options:

l, --language INDEX
Set the language to be shown by index number: [0|en_US.UTF-8]: English, [1|zh_TW.BIG5]: Traditional Chinese (Big5) - Taiwan, [2|zh_TW.UTF-8]: Traditional Chinese (UTF-8, Unicode) - Taiwan [a | ask]: Prompt to ask the language index

-b, -batch, --batch
(DANGEROUS!) Run program in batch mode, i.e. without any prompt or wait for pressing enter key. //NOTE// You have to use -batch instead of -b when you want to use it in the boot parameters. Otherwise the program init on system will honor -b, too.

-c, --confirm
Wait for confirmation before saving or restoring

-d, --debug-mode
Enter command mode to debug before saving/restoring

--debug=LEVEL
Output the partimage debug log in directory /var/log/ with debug LEVEL (0,1,2... default=0)

-m, --module MODULE
Force to load kernel module MODULE, this is useful when some SCSI device is not detected. NOTE! Use only one module, more than one may cause parsing problem.

-o0, --run-prerun-dir
Run the script in the directory /usr/share/drbl/postrun/ocs/ before clone is started. The command will be run before MBR is created or saved.

-o1, -o, --run-postrun-dir
Run the script in the directory /usr/share/drbl/postrun/ocs/ when clone is finished. The command will be run before that assigned in -p or --postaction.

-w, --wait-time TIME
Wait for TIME secs before saving/restoring

-nogui, --nogui
Do not show GUI (TUI) of Partclone or Partimage, use text only
-a, --no-force-dma-on
   Do not force to turn on HD DMA

-mp, --mount-point MOUNT_POINT
   Use NFS to mount MOUNT_POINT as directory ocsroot (ocsroot is assigned in drbl.conf)

-or, --ocsroot DIR
   Specify DIR (absolute path) as directory ocsroot (i.e. overwrite the ocsroot assigned in drbl.conf)

-p, --postaction [choose | poweroff | reboot | command | CMD]
   When save/restoration finishes, choose action in the client, poweroff, reboot (default), in command prompt or run CMD

-ns, --ntfs-progress-in-image-dir
   Save the ntfsclone progress tmp file in the image dir so that if cloning is in DRBL client, the progress can be check in the server (Default in to be put in local /tmp/, which is local tmpfs).

-um, --user-mode [beginner | expert]
   Specify the mode to use. If not specified, default mode is for a beginner.

-v, --verbose
   Prints verbose information

-d0, --dialog
   Use dialog

-d1, --Xdialog
   Use Xdialog

-d2, --whiptail
   Use whiptail

-d3, --gdialog
   Use gdialg

-d4, --kdialog
   Use kdialog

-x, --interactive
   Interactive mode to save or restore.

Example:

- To save or restore image in client (Only that DRBL client will join, and its local partitions is NOT mounted). NOTE!!! You should run the command in DRBL client or you have to make sure the target device is NOT busy!. To save all the data in local first IDE harddrive hda as image IMAGE1,
use ntfsclone instead of partimage, and lzop compression (NOTE!!! You should run the command in DRBL client or make sure hda is NOT busy/mounted!): ocs-sr --use-ntfsclone -z3 savedisk IMAGE1 hda

- To save the data in first and second partitions in local first IDE harddrive hda as image IMAGE2, use ntfsclone instead of partimage, and lzop compression (NOTE!!! You should run the command in DRBL client, or make sure hda is NOT busy/mounted!): ocs-sr --use-ntfsclone -z3 saveparts IMAGE2 "hda1 hda2"

- To restore image IMAGE1 to local hda. grub-install will be run after cloning (image IMAGE1 is already in DRBL server. NOTE!!! You should run the command in DRBL client or make sure hda is NOT busy/mounted!): ocs-sr -g auto restoredisk IMAGE1 hda

- To restore image first and second partitions from IMAGE2 to local hda1 and hda2. grub-install will be run after cloning (image IMAGE2 is already in DRBL server. NOTE!!! You should run the command in DRBL client or make sure hda is NOT busy/mounted!): ocs-sr -g auto restoreparts IMAGE2 "hda1 hda2"

- To save disk(s)/partition(s) as an image or restore an image to disk(s)/partition(s) interactively, use: ocs-sr -x

disk-to-disk Operation


Clone disk or partition on-the-fly

The "ocs-onthefly" is used to do disk to disk or partition to partition copy on-the-fly. This command is different from drbl-ocs (or clonezilla). Clonezilla is used to do massively clone, so it will save the template machine as an image in clonezilla server. On the other hand, ocs-onthefly is used to 1 to 1 copy, so no image will be saved in the server. Just clone disk or partition directly.

There are 2 ways to run ocs-onthefly:

1. Clone locally: Boot the machine as DRBL client, then clone one disk to another disk. This is specially for when you just want to clone disk, and you only have one machine.

2. Clone via network: Boot the source and target machine as DRBL clients, then clone disk from one machine to another machine. This is specially for you have 2 machines, and you want to clone them without dismantling machine.

Usage:
ocs-onthefly [OPTION]
Option:
-e, --resize-partition
    resize the target disk in target machine (To solve the small partition image restored to larger partition problem.)
-f, --source DEV
    specify the source device as DEV (hda, hda1...)

-g, --grub-install GRUB_PARTITION
    install grub in hda with root grub directory in GRUB_PARTITION when restoration finishes, GRUB_PARTITION can be one of "/dev/hda1", "/dev/hda2"... or "auto" ("auto" will clonezilla detects the grub root partition automatically)

-i, --filter PROGRAM
    use the PROGRAM (gzip/lzop/bzip2/cat) before sending partition data to netcat (only in network clone mode). The default action is gzip. Use "cat" if you do not want to compress (Good for fast internode network).

-n, --no-sfdisk
    skip partition table creation

-m, --no-mbr-clone
    do NOT clone MBR

-o, --load-geometry
    force to use the saved CHS (cylinders, heads, sectors) when using sfdisk in restoring.

-p, --port PORT
    specify the netcat port (Only in network clone mode)

-r, --server
    specify the running machine is in network clone server.

-s, --source-IP
    IP specify the source IP address (used in target client machine).

-t, --target DEV
    specify the target device as DEV (hda, hda1...)

-v, --verbose
    prints verbose information

ocs-onthefly [OPTION]

Examples:

1. Clone locally: To clone the 1st harddisk (hda) to 2nd harddisk (hdb), you can boot this machine as DRBL client, then run:
   ocs-onthefly -f hda -t hdb

2. Clone via network: To clone the 1st harddisk (hda) in machine A to the 1st harddisk (hda) in machine B. Then without dismantling machines, you can do it by:
Boot machine A as DRBL client, and its IP address is, say, 192.168.100.1, then run:
```
ocs-onthefly -r -f hda
```
Then it will prompt you the command to run in machine B, such as:
```
ocs-thefly --source-IP 192.168.100.1 -t [TARGET_DEV] (TARGET_DEV is like hda, hdb, hda1, hdb1...)
```
The "TARGET_DEV" is the target harddisk in machine B, in this case, it hda. Then, boot machine B as DRBL client, and run:
```
ocs-onthefly --source-IP 192.168.100.1 -t hda
```

### 9.15. opsi-server with multiple depots (free)

#### 9.15.1. Concept

Supporting multiple depot shares in opsi aims at the following targets:

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

Accordingly, it is implemented:

- All configuration data is stored on the central `opsi-config-server`.
- All clients connect to this `opsi-config-server` in order to request their configuration data. The configuration data comprise the information on method and target of the `opsi-depot-server` connection.
- All installable software is stored on `opsi-depot-servers`.
- The `opsi-depot-servers` have as well an `opsipxeconfd` running by which they provide boot-images to clients via PXE/tftp.
- `opsi-package-manager`
  A program to (de-)install opsi packages on one ore more `opsi-depot-servers`.
- The opsi packages are copied via webdav protocol to the `opsi-depot-servers` and are installed from the `opsiconfd` via a web service call.
- `opsi-configed` supports the management of multiple depots.
- Clients connected to different depots can be managed in one bundle if the involved depots are synchronized (have all product packages in identical versions).

The following schema gives a more detailed view on the communication between the components of a opsi multi depot share environment.
Figure 148. Scheme: opsi config server without attached depot server (single location)
In order to create an *opsi-depot-server* you have to install a standard *opsi-server*. This *opsi-server* can be configured to act as *opsi-depot-server* by calling the script `opsi-setup --register-depot` as user root on the server which should become the *opsi-depot-server*. Because this script does not only reconfigure the local server, but also registers this server as *opsi-depot-server* with the central *opsi-config-server*, username and password of a member of the *opsiadmin* group have to be supplied here.

On Univention Corporate Server the registration of a *opsi-depot-server* happens automatically. The
first server with an opsi installation is used as **opsi-config-server** and all following in a UCS domain installed systems will register there as an **opsi-depot-server**.

Example: 
**svmdepotde.svm.local** will be reconfigured as opsi-depot-server and registered at the opsi-config-server **sepiella.svm.local**:

```
root@svmdpotde.svm.local:~# opsi-setup --register-depot
```

Now you will be prompted for the opsi-config-server you want to connect to. The registration needs to be authorised by supplying the username and password of a member of the group **opsiadmin** at the opsi-config-server.

![Figure 150. opsi-setup --register-depot: Enter opsiadmin account for the opsi-config-server](image)

Now the opsi-depot-server settings are being displayed. In most cases no changes have to be made. Note that the new opsi-depot-server will register as a "Master-Depot", so that you'll be able to assign **opsi-clients** to it.
After the data input is completed the configuration process will start:
Usually the configuration files in `/etc/opsi/package-updater.repos.d/` on the new depot should be checked.

If the new depot should only update its packages from the main server, only the repository in `/etc/opsi/package-updater.repos.d/opsi-server.repo` should remain active. A possible configuration can look like this:
Non-interactive registration of a opsi-depot-server

Since opsi-depotserver 4.0.7.2 it is possible to register a depot without interaction.

To do this the data for the connection to the opsi-config-server has to be passed as JSON object alongside the parameter --unattended.

```
opsi-setup --register-depot --unattended '{"address": "config.server.address:4447/rpc", "username": "adminuserinopsi", "password": "pwoftheuser"}'
```

The opsi-depot-server will be created with defaults.

It is possible to set custom attributes for the opsi-depot-server. For this the JSON object needs to get the key depot and as a value another JSON object with the custom values.

The following example illustrates how to set a custom description:

```
opsi-setup --register-depot --unattended '{"address": "config.server.address:4447/rpc", "username": "adminuserinopsi", "password": "pwoftheuser", "depot": {"description": "Added with unattended registration."}}'
```

9.15.3. package management with multiple depots

see also:
Section 5.3.2, “Tool: opsi-package-manager - (de-)install opsi-packages”
Section 5.3.3, “Tool: opsi-package-updater”

In or to manage opsi-packages with different opsi-depot-server the opsi-package-manager got the option -d (or --depot). With this option you can give the target opsi-depot-server for the installation. Using the keyword ALL the opsi package will be copied to /var/lib/opsi/repository on all known opsi-depot-servers and then installed via a web service call.

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

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

```
opsi-package-manager -d ALL -i softprod_1.0-5.opsi
```

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

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

```
opsi-package-manager -D -d ALL mshotfix
mshotfix
  vmix12.uib.local : 200804-1
  vmix13.uib.local : 200804-1
  bonifax.uib.local: 200805-2
```

### 9.16. Dynamic Depot Assignment (free)

#### 9.16.1. Introduction

With the standard multi-depot support in opsi, the clients are permanently assigned to the respective depots. This is now enhanced by a mechanism with which a client can detect from which depot it can obtain the software the fastest.

For most cases an assignment according to the IP address might be the easiest and most suitable solution. For other network topologies, e.g. a star topology VPN network, this might not be sufficient.

Therefore a mechanism is required for the client to dynamically detect, which depot to connect for the download of software packages. The algorithm and implementation depends on the network topology and other special customer requirements. So it is best to have this adaptable and configurable.

To offer the option, that the client can detect the suitable depot according to the current network conditions, it must be ensured, that the alternative depots are synchronized, which means they offer the same software packages. In practice the depots will not be synchronized at all times. So the list depots offered to a client is limited to those depots, which are synchronized with the master depot of the client. The master depot of a client is the depot, to which the client is assigned to. The master depot thus determines which software and which version can be installed on the client.

Our concept for this is as follows:

The opsi configserver provides a client script, which is transferred to the client and interpreted there if necessary. This script determines which of the available depots is used. For this client script is defined: the interface with which the script gets the list of available servers and the current client configuration (IP address, netmask, gateway) and to return the result of the selection procedure.
Furthermore there are interfaces for logging and information about the ongoing process.

The specific implementation of this script can easily be adapted to the requirements of the particular opsi environment.

Resulting from this concept, the sequence of a client connection then looks as follows:

1. The client reports to the opsi configserver via the web service.
2. The opsi configserver sends the client the list of software packages to be installed.
3. The opsi-configserver transmits to the client the script for detecting the best depot and the list of available depots.
4. The client executes the script and uses it to determine the best depot.
5. The client connects to the selected depot to get the required software packages.
6. The installation status is reported back to the opsi configserver.

9.16.2. Requirements

The co-financing process of this module was completed in March 2013. Further details can be found in Section 9.1, “Activation of non-free modules”.

The following package versions are required:

<table>
<thead>
<tr>
<th>opsi package</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-client-agent</td>
<td>&gt;=4.0.11</td>
</tr>
<tr>
<td>opsi-configed</td>
<td>&gt;=4.0.1.5-1</td>
</tr>
<tr>
<td>python-opsi</td>
<td>&gt;=4.0.0.18-1</td>
</tr>
</tbody>
</table>

The depot selection is realized via the opsi-client-agent. This means that this function is not available with netboot products.

9.16.3. Configuration

The script that the client uses to select the depot, is stored on the server as the following file:

/etc/opsi/backendManager/extend.d/70_dynamic_depot.conf

To activate the dynamic depot selection for a client, the following host parameter has to be set: clientconfig.depot.dynamic = true

This can be done via opsi configed in the host parameters tab.

This can also be done on the command line with the command opsi-admin (client-id> has to be replaced by the FQDN of the client, e.g. client1.uib.local):
opsi-admin -d method configState_create clientconfig.depot.dynamic <client-id> [True]

The result can be checked by executing:

```bash
opsi-admin -d method configState_getObjects []
'{"configId":"clientconfig.depot.dynamic","objectId":"<client-id>"}'
```

### 9.16.4. Editing depot properties

The properties of a depot are partly determined when an opsi-server is registered as a depot with the command `opsi-setup --register-depot` (see Section 9.15.2, “Creating an depot server”).

You can edit the depot properties later. This can be done from the management interface as well as on the command line.

![Figure 152. Showing the properties of a depot (2nd button from the left)](image)

You can call up the depot properties using the button `Properties of depots` in the top right of the management interface.

![Figure 153. Depot properties in opsi-configed](image)
On the command line the depot properties can be shown with the method `host_getObjects`. Here e.g. for the depot `dep1.uib.local`.

```
opsi-admin -d method host_getObjects [] '{"id":"dep1.uib.local"}'
```

This example results in the following output:

```
[
  {
    "masterDepotId": "masterdepot.uib.local",
    "ident": "dep1.uib.local",
    "networkAddress": "192.168.101.0/255.255.255.0",
    "description": "Depot 1 Master Depot",
    "inventoryNumber": "",
    "ipAddress": "192.168.105.1",
    "repositoryRemoteUrl": "webdavs://dep1.uib.local:4447/repository",
    "depotLocalUrl": "file:///var/lib/opsi/depot",
    "isMasterDepot": true,
    "notes": "",
    "hardwareAddress": "52:54:00:37:c6:8b",
    "maxBandwidth": 0,
    "repositoryLocalUrl": "file:///var/lib/opsi/repository",
    "opsiHostKey": "6a13da751fe76b9298f4ede127280809",
    "type": "OpsiDepotserver",
    "id": "dep1.uib.local",
    "depotWebdavUrl": "webdavs://dep1.uib.local:4447/depot",
    "depotRemoteUrl": "smb://dep1/opsi_depot"
  }
]
```

To edit the depot properties from the command line, the output is written to a file:

```
opsi-admin -d method host_getObjects [] '{"id":"dep1.uib.local"}' > /tmp/depot_config.json
```

The resulting file (`/tmp/depot_config.json`) can now be edited and written back with the following command:

```
opsi-admin -d method host_createObjects < /tmp/depot_config.json
```

The depot properties, which are relevant in the context of dynamic depot allocation are:

- **isMasterDepot**
  
  Must be **true** for assigning a client to this depot. If **false** is entered here, no clients can be assigned,
but the depot is still used in the dynamic depot assignment.

- **networkAddress**
  Network addresses for which this depot is responsible. The network address can be specified in two notations:
  - network/netmask, example: 192.168.101.0/255.255.255.0
  - network/maskbits, example: 192.168.101.0/24

Whether the *networkAddress* is actually evaluated to determine the depot depends on the algorithm in the selection script. The default algorithm supplied by uib uses this.

### 9.16.5. Synchronization of depots

To keep the depots in sync, opsi provides several tools:

- **opsi-package-manager**
- **opsi-package-updater**

When installing an opsi package, the **opsi-package-manager** can be instructed to install the package not only on the current server but on all known depots by using the parameter `-d ALL`. Example:

```
opsi-package-manager -i opsi-template_1.0-20.opsi -d ALL
```

By using the parameter `-D`, **opsi-package-manager** can be instructed to list the differences between depots. A list of depots must be specified with the `-d` option or all known depots must be selected with `-d ALL`. Example:

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

**opsi-package-manager** is also the tool used for a *push* synchronization. On the other hand, the tool **opsi-package-updater** is intended to synchronize depots using the *pull* method.

**opsi-package-updater** can run on the depots as a cronjob. This enables easy automation. Please refer to the chapter **Section 5.3.3, “Tool: opsi-package-updater”** for further information on the configuration.

If a package is installed on an opsi-server with **opsi-package-manager -i** (without `-d`), it does not end up in the repository directory. In order for it to be copied there, you can either specify the name of the depot during installation with `-d` or explicitly instruct the upload to the repository directory with **opsi-package-manager -u <name of the package>**.

Please also note the description of the two tools in the corresponding chapters of the opsi manual.
9.16.6. Processing

If the dynamic depot assignment is activated for a client via the host parameter `clientconfig.depot.dynamic`, the client retrieves the script via the web service and executes it.

The script that the client uses to select the depot is stored on the server as the file:

/etc/opsi/backendManager/extend.d/70_dynamic_depot.conf

The following parameters are transferred to the `selectDepot` function defined in this script:

- **clientConfig**
  Information of the current client configuration (hash).
  The `clientConfig` hash keys are currently:
  - "clientId": opsi host ID of the client (FQDN)
  - "ipAddress": IP address of the network interface used to connect to the configserver
  - "netmask": network mask of the network interface
  - "defaultGateway": default gateway

- **masterDepot**
  Information about the master depot (`opsi-depot-server`-object). The master depot is the depot to which the client is assigned in the management interface. The attributes of the transferred `opsi-depot-server`-object correspond to the attributes as given by `host_getObjects` (see Section 9.16.4, “Editing depot properties”).

- **alternativeDepots**
  Information about the alternative depots (list of `opsi-depot-server`-objects). The list of alternative depots is determined from the depots which are synchronised to the master depot, in regard to the software packages currently required.

Based on this information, the algorithm can now select a depot from the list. The `opsi-depot-server`-object of the depot to be used must be returned by the function. If the algorithm does not find a suitable depot from the list of alternative depots or if this is empty, the master depot should be returned.

9.16.7. Selection script template

Three functions for selecting a depot are already implemented in the template script.

The function `depotSelectionAlgorithmByNetworkAddress` checks the network addresses of the depots and selects the depot where the current IP address is in the network of the depot.

The function `depotSelectionAlgorithmByLatency` sends ICMP echo request packets (ping) to depots and selects the depot with the lowest latency.

The function `depotSelectionAlgorithmByMasterDepotAndLatency` is intended for environments with several master depots, which can have slave depots assigned. From the set of master depots of the client and the associated slave depots, the depot that has the lowest latency is selected.

The function `getDepotSelectionAlgorithm` is called by the client and returns the algorithm to be used for the selection of the depot. Without changes to the template script, the function
```python
# -*- coding: utf-8 -*-

global depotSelectionAlgorithmByNetworkAddress
depotSelectionAlgorithmByNetworkAddress = 
'
''

def selectDepot(clientConfig, masterDepot, alternativeDepots=[]):
    selectedDepot = masterDepot
    logger.info(u"Choosing depot from list of depots:")
    logger.info(u"  Master depot: %s % masterDepot")
    for alternativeDepot in alternativeDepots:
        logger.info(u"  Alternative depot: %s % alternativeDepot")
    if alternativeDepots:
        import socket, struct
        # Calculate bitmask of host's ipaddress
        n = clientConfig['ipAddress'].split('.')
        for i in range(4):
            n[i] = forceInt(n[i])

depots = [ masterDepot ]
depots.extend(alternativeDepots)
for depot in depots:
    if not depot.networkAddress:
        logger.warning(u"Network address of depot '%s' not known" % depot)
        continue
    (network, netmask) = depot.networkAddress.split('/')
    while (network.count('.') < 3):
        network = network + u'.0'
    if (netmask.find('.') == -1):
        netmask = forceUnicode(socket.inet_ntoa(struct.pack('>I',0xffffffff ^ (1 << 32 - forceInt(netmask)) - 1)))
        while (netmask.count('.') < 3):
            netmask = netmask + u'.0'

    logger.debug(u"Testing if ip %s is part of network %s/%s" %
                  (clientConfig['ipAddress'], network, netmask))

    n = network.split('.')
    for i in range(4):
        n[i] = int(n[i])
    n = netmask.split('.')
    for i in range(4):
        n[i] = int(n[i])
```

`depotSelectionAlgorithmByNetworkAddress` will be returned.
wildcard = netmask ^ 0xFFFFFFFFL
if (wildcard | ip == wildcard | network):
    logger.notice(u"Choosing depot with networkAddress %s for ip %s" %
    (depot.networkAddress, clientConfig['ipAddress']))
    selectedDepot = depot
    break
else:
    logger.info(u"IP %s does not match networkAddress %s of depot %s" %
    (clientConfig['ipAddress'], depot.networkAddress, depot))
return selectedDepot


global depotSelectionAlgorithmByLatency
depotSelectionAlgorithmByLatency = \


def selectDepot(clientConfig, masterDepot, alternativeDepots=[]):
    selectedDepot = masterDepot
    logger.info(u"Choosing depot from list of depots:")
    logger.info(u"   Master depot: %s" % masterDepot)
    for alternativeDepot in alternativeDepots:
        logger.info(u"   Alternative depot: %s" % alternativeDepot)
    if alternativeDepots:
        from OPSI.Util.Ping import ping
        from OPSI.Util.HTTP import urlsplit
        depots = [ masterDepot ]
        depots.extend(alternativeDepots)
        latency = {}
        for depot in depots:
            if not depot.repositoryRemoteUrl:
                continue
            try:
                (scheme, host, port, baseurl, username, password) =
                urlsplit(depot.repositoryRemoteUrl)
                latency[depot] = ping(host)
                logger.info(u"Latency of depot %s: %0.3f ms" % (depot,
                latency[depot]*1000))
            except Exception, e:
                logger.warning(e)
            if latency:
                minValue = 1000
                for (depot, value) in latency.items():
                    if (value < minValue):
                        minValue = value
                        selectedDepot = depot
                logger.notice(u"Choosing depot %s with minimum latency %0.3f ms" %
                (selectedDepot, minValue*1000))
    return selectedDepot
global depotSelectionAlgorithmByMasterDepotAndLatency
depotSelectionAlgorithmByMasterDepotAndLatency = \
```python
def selectDepot(clientConfig, masterDepot, alternativeDepots=[]):
    def getLatencyInformation(depots):
        from OPSI.Util.Ping import ping
        from OPSI.Util.HTTP import urlsplit

        latency = {}
        for depot in depots:
            if not depot.repositoryRemoteUrl:
                continue

            try:
                (scheme, host, port, baseurl, username, password) = urlsplit(depot.repositoryRemoteUrl)
                latency[depot] = ping(host)

            except Exception, e:
                logger.warning(e)
        return latency

    def getDepotWithLowestLatency(latency):
        selectedDepot = None
        if latency:
            minValue = 1000
            for (depot, value) in latency.items():
                if not value:
                    continue
                if (value < minValue):
                    minValue = value
                    selectedDepot = depot

            logger.notice(u"Choosing depot %s with minimum latency %0.3f ms" % (selectedDepot, minValue*1000))
        return selectedDepot

    logger.info(u"Choosing depot from list of depots:")
    logger.info(u"   Master depot: %s" % masterDepot)
    for alternativeDepot in alternativeDepots:
        logger.info(u"   Alternative depot: %s" % alternativeDepot)
    if alternativeDepots:
        from collections import defaultdict
```
def getDepotSelectionAlgorithm(self):
    #return depotSelectionAlgorithmByMasterDepotAndLatency
    #return depotSelectionAlgorithmByLatency
    return depotSelectionAlgorithmByNetworkAddress

9.16.8. Logging

If the dynamic depot assignment is activated, the corresponding entries from the depot selection can be found in opsiclientd.log. Here is the shortened log of an example session. In this example the server bonifax.uib.local is the configserver and master depot for the client pctry4detlef.uib.local. The master server bonifax.uib.local has the network address 192.168.1.0/255.255.255.0. Available as an alternative depot is stb-40-srv-001.uib.local with the network address 192.168.2.0/255.255.255.0. The client pctry4detlef.uib.local has the IP address 192.168.2.109, and is therefore in the network of the alternative depot.
9.17. opsi Software On Demand - opsi-client-kiosk (free)
9.17.1. Introduction

With the module "Software-on-Demand" opsi administrators may give their users access to install a range of software-products. These software products may be selected and installed user-driven without the administrator needing to do anything. This documentation shows how the module "Software-on-Demand" works, describes its functions and how to configure it.

Conventions: When the opsi-client-kiosk/agent in general is mentioned, it refers to all OS versions (Windows/Linux) unless stated otherwise. When talking about packages or products then the opsi-client-kiosk/agent package/product is the Windows package/product and the l-opsi-client-kiosk or opsi-linux-client-agent package/product is the Linux package/product.

9.17.2. Prerequisites

There are some preconditions needed for the usage of this extension. The product-groups are needed, available with opsi 4.0. Furthermore the opsi-client-agent and the opsi-configed at version 4.0.1 are needed.

Windows: Starting with opsi-client-agent version 4.1.1.6-5 the kiosk is available as a standalone package (opsi-client-kiosk). It will be however installed during the installation of the opsi-client-agent if it's not already present. For more details please check New opsi-client-kiosk application

Table 31. Required Packages

<table>
<thead>
<tr>
<th>opsi-package</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-client-agent / opsi-linux-client-agent</td>
<td>&gt;=4.1.0.0-36, &gt;=4.1.1.6-5** / &gt;= 4.1.1.11-1</td>
</tr>
<tr>
<td>opsi-winst / opsi-script</td>
<td>&gt;=4.10.8.12</td>
</tr>
<tr>
<td>python-opsi</td>
<td>&gt;=4.1.1.71-1</td>
</tr>
<tr>
<td>opsi-server</td>
<td>&gt;=4.0.1-2</td>
</tr>
<tr>
<td>opsi-configed</td>
<td>&gt;=4.0.1.6-1</td>
</tr>
<tr>
<td>(l-)opsi-client-kiosk</td>
<td>&gt;=4.1.1.7-6</td>
</tr>
</tbody>
</table>

**opsi-client-kiosk as a standalone package

9.17.3. Configuration

The configuration of this extension is based on product-groups and config-variables. The used config-variables are:

- software-on-demand.active
- software-on-demand.product-group-ids
- software-on-demand.admin-mode
• software-on-demand.installation-now-button

These config-variables are created with installing the opsi-server-package.

Starting from Version >= 4.1.1.11 the kiosk uses the caching and synchronization methods of the WAN/VPN-client if running on a WAN/VPN-client. Thus the kiosk can be used on a WAN/VPN-client too. Therefore the config-variable **software-on-demand.installation-now-button** has to be set to false. Otherwise display and messages of the kiosk are not synchronize with the current client status.

**Managing product-groups**

The most comfortable way to create and manage product-groups is using the **opi-configed**. There you have to change to the tab **product configuration**.

Since version 4.0.1.6 of the **opi-configed** you can change to **product configuration** without choosing a client.

The product-group menu is above the product list.

![Figure 154. product-group menu](image)

With the drop down menu you can choose a product-group to edit it. If you have chosen a group, the corresponding products will be highlighted.

With a second icon, filter can be activated or deactivated. When a filter was activated, only the products of the activated product-group are seen.

Product-groups can be edited after activating the icon with the yellow packets (show editor / hide editor) next to the icon with the filter. In this view, a new group and it’s description can be added.

Save the editing by activating the red check icon.

If some more products should be added to a group, select them and press the red check icon. (Press the <ctrl> button and select the products).
configure the module Software-On-Demand

The module can be configured, as mentioned above, with the config-variables described in the following table:

Table 32. overview of the config-variables of the module Software-on-Demand

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>software-on-demand.active</td>
<td>activates or deactivates the module, is used as a flag for the migration (<em>Nur Windows: Installation über einen alten opsi-client-agent der den alten Kiosk noch enthält</em>). It’s irrelevant for the opsi-client-kiosk with version &gt;=4.1.1.6-5.</td>
<td>true/false</td>
</tr>
<tr>
<td>software-on-demand.product-group-ids</td>
<td>Product-groups with software-products, that can be used for Software-on-Demand.</td>
<td>List of product-groups</td>
</tr>
<tr>
<td>software-on-demand.admin-mode</td>
<td>Enables specific functionality for administrative purpose/setting icons/screenshots. Should be enabled only if needed (client-specific)</td>
<td>true/false</td>
</tr>
<tr>
<td>software-on-demand.installation-now-button</td>
<td>Enables to install products immediately. Do NOT use on WAN/VPN-clients</td>
<td>true/false</td>
</tr>
</tbody>
</table>

There are two ways to use these configuration objects. For the whole system or for each client. The following 2 chapters will explain both ways.

Using the kiosk (Version>= 4.1.1.11) on a WAN/VPN-client the config-variable `software-on-demand.installation-now-button` has to be set to `false`. Otherwise display and messages of the kiosk are not synchronize with the current client status.

Configuration for the whole system

The configuration here is the default system wide for every client. The configuration can be edited in the `opsi-configed`. Push the Button `Server Configuration` and change to the tab `Host Parameter`
Figure 155. part of the module server configuration in the opsi configuration editor

Another possibility is to change the server-configuration with the following command:

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

Figure 156. edit-config-defaults with opsi-setup

Administration is also possible with the opsi-python-API or with `opsi-admin`

Configuration for a single client

The configuration for a single client - or client specific configuration - is useful if, for example, only some of the clients should have access to the Software-on-Demand extension. Or if you want to make
several product groups available to some clients.

The configuration of the client specific host parameters can be edited in different ways:

The most comfortable way to edit the configuration is via `opsi-configed`. Choose one or several clients (even all clients of a client group if you want to) and then navigate to the tab "Host parameters".

![Figure 157. edit hostparameter in the configuration editor](image)

This editing overrides the system wide defaults.

### 9.17.4. New opsi-client-kiosk application

#### Windows

With the opsi-client-agent version (>=4.1.1.6-5) the kiosk is available as a standalone product (`opsi-client-kiosk`) and will be installed as such when installing the opsi-client-agent. It tries to migrate the existing kiosk settings.

Background of this changes are:

- simpler maintenance of the opsi-client-agent package
- Changes to the kiosk can be published independently from the opsi-client-agent package
- Allows to apply new settings of the kiosk to the clients without the need to reinstall the opsi-client-agent
When installing the opsi-client-agent (>=4.1.1.6-5) on the depot, the opsi-client-kiosk package will be installed as well if necessary. For clients, when installing the opsi-client-agent (>=4.1.1.18-1) on the afore mentioned, the opsi-client-kiosk product is set to setup if necessary. For details see Installation über einen alten opsi-client-agent der den alten Kiosk noch enthält.

The old (web based) Kiosk client no longer works with the new opsi-client-agent/opsiclientd.

Linux

The opsi-client-kiosk is also available under Linux as l-opsi-client-kiosk package or product.

opsi-client-kiosk: Installation on a depot

If you invested time into customizing the kiosk, it's recommended to backup the custom directory before installing a new opsi-client-kiosk/opsi-client-agent on the depot:

Kiosk as a standalone package (opsi-client-kiosk or l-opsi-client-kiosk): /var/lib/opsi/depot/(l-)opsi-client-kiosk/files/app/ock_custom


In rare cases also /var/lib/opsi/depot/opsi-client-agent/files/opsiclinetkiosk/files_to_copy/ock_custom

First installation / installation over an already existing (l-)opsi-client-kiosk product

Install the current (l-)opsi-client-kiosk package (available at download.uib.de) on the desired depot and update the opsi-(linux)-client-agent package if needed.

opsi-client-agent package >=4.1.1.6-5 had to be installed on the depot as well as on the respective client.

Windows only: Installation through an old opsi-client-agent which still contains the old kiosk (i.e. contains the directory opsiclientkiosk, or <=4.1.0.0)

To minimize the number of manual changes or interactions, it is attempt to migrate the kiosk settings automatically from the opsi-client-agent to the opsi-client-kiosk package and install it on the depot or the corresponding clients if needed. Usually this is done automatically without the need of further interaction.

For a better understanding of the process described here in more detail, it's important to distinguish what happens during the installation of the opsi-client-agent on a depot and what happens during the installation on a client.

Installation of the opsi-client-agent Version (>=4.1.1.6-5) on a Depot
When installing opsi-client-agent on the depot, the opsi-client-kiosk package is installed on the depot if certain conditions are met. It tries to migrate the kiosk settings from the opsi-client-agent package to the opsi-client-kiosk package. This happens only if the following conditions are fulfilled:

- the opsi-client-kiosk package is NOT installed on the depot.
- the opsi-client-agent (>=4.1.1.6-5) is installed over an old opsi-client-agent that still contains the old kiosk (i.e. contains the directory opsiclientkiosk, <=4.1.0.0)

**Installation of the opsi-client-agent Version (>=4.1.1.18-1) on a Client**

When installing the opsi-client-agent (>=4.1.1.18-1) on a client, the opsi-client-kiosk is set to setup under certain conditions. This happens when the following conditions are met:

- the opsi-client-kiosk is NOT installed on the client
- the config/host-parameter software-on-demand.active is set to true for this client

So in most cases the opsi-client-kiosk will be installed on the client automatically after the opsi-client-agent installation, due to the fact that the client PC will be rebooted and the opsi-clientd event gui_startup will be triggered which should be configured in most cases.

If the kiosk is not available on some clients, despite attempted automatic migration/installation, please check:

1. if an opsi-client-kiosk package is installed on the depot the client is connected to, if not install the opsi-client-kiosk package or the current opsi-client-agent package, respectively
2. if the opsi-client-kiosk is installed on the client, should that not be the case please set it to setup

**opsi-client-kiosk: Installation on a client**

The installation on a client is controlled by the following `opsi-client-kiosk` product properties:

- **startmenu_entry**
  The Start Menu entry for the opsi-client-kiosk (software on demand).
  Default=`software on demand`; editable

- **startmenu_folder**
  The Start Menu folder for the opsi-client-kiosk (opsi.org)
  Default=`opsi.org`; editable

- **desktop_icon**
  Create a desktop icon for opsi-client-kiosk?
  Default=`false`

- **install_icon_collection**
  For administrative purposes an icon collection can also be installed. It is only recommended though if product icons are to be set on the client via the kiosk admin mode and you want to select some of them from the icon collection.
  Default=`false`
The used icon for the start menu / Desktop may be modified by storing a `kiosk.ico` file (Windows) or `kiosk.png` file (Linux) at `/var/lib/opsi/depot/opsi-client-kiosk/files/app/ock_custom/skin/` (since opsi-client-kiosk version 4.1.1.7-2).

**opsi-client-kiosk: Usage**

**Standard mode**

**Main window (standard mode)**

After start of the kiosk all products which are assigned to the kiosk are shown on product tiles within the main window. The switch "All" is marked in the filter switch panel (Figure 5).

![Softwarekorb](image.png)

**Figure 158.** Kiosk (standard mode) - Main window with product tiles.  (1) Window panel. (2) Header panel. (3) Filter switch. (4) Opens the search mask. (4) Opens the help. (6) product tile

**Window panel (1) and Header panel (2)**

In the standard mode the version of the kiosk and, if applicable, the info that installations as from now are disabled are shown in the window panel (1) (see also dialogs for installation/uninstallation and update of the product). The header panel (2) is customisable. See section Corporate Identity of the opsi-client-agent for more details.
Filter switch (3)
By clicking on the filter switch (3) "Updates", "Not Installed" or "Actions" only products are shown with the chosen status.

Product tiles (4)
Products are displayed by product tiles. Product tiles show first information about the product. That are the name of the product, if set, a specific product icon otherwise a standard icon, as well as the status of the product and, if a action is set, the respective action.

Only if an action is set an action is displayed otherwise nothing is displayed.

After clicking on a product tile the tiles are hidden and detailed information of the selected product are shown (Figure 7).

Search (5)
Click on the magnifying glass to open a search field (Figure 6).

![Search field of the kiosk](image)

Use the search field to search for specific products. All fields of the product will be searched. Only products which fit to the search term will be displayed e.g. because the term is in the name or in the description of the product. Clicking on th "X" within the search field will clear the search field and all products are displayed again. The same effect has a click on the magnifying glass above the search filed (Figure 5) but additionally the search field is hidden.

Help (6)
Via the button "help" the help window is reached.

Detailed product view
After clicking on a product tile the detailed product view is shown.
Figure 160. Kiosk (standard mode) - detailed product view. (1) back to main window. (2) (Un)install product respectively removes action. (3) Info block (5) Updates the product. (6) Description and hints of the product.

**Back (1)**
Back to main window (Figure 5).

**Install/Uninstall product or removes action (2)**
Clicking on this button opens a dialog to install/uninstall the product or removes the action set before.

- The functionality of this button depends on the product state (installed/not installed, action set)

**Info block (3)**
The info block shows the installed version and the current available version as well as the respective action if it is set.

**Screenshot (4)**
Shows, if available, a screenshot of the product.

- Ask your system administrator if no screenshot is displayed but you like to have one.
Update (5)
Opens a dialog to update the product.

This button is only displayed if an update is available i.e. the installed version differs from the newest available version.

Description and Notes (6)
Here the product description and notes are presented.

The product description and notes are only as good as they are incorporated into the opsi-products. Ask your system administrator or package manager if they are not understandable.

Dialogs for installation/uninstallation and updating of products
A dialog is displayed after clicking on the respective button within the detailed product view. Either you are directly prompted for confirmation of the action (Figure 8) or an additional dialog is shown where you can select the starting point of the requested action (now or later after standard event e.g. reboot, Figure 9)

Figure 161. Kiosk - Dialog for confirmation of the requested action.

Figure 162. Kiosk - Dialog to select the starting point of the requested action.

The dialog for selecting the starting point of the requested action is only shown if the installation as from now is not disabled by your system administrator.

Help window
The help window (Figure 8) displays information about the opsi client kiosk. Via the help window it is also possible to launch this manual. In addition the kiosk can be put into the expert mode from here.
9. opsi Extensions

Figure 163. Kiosk - Help window. (1) Info block. (2) Opsi manual. (3) Check box for expert mode

Info block (1)
Here information about the opsi client kiosk are shown. Via the links it is possible to launch the uib or opsi webpage in your preferred browser.

Opsi manual (2)
Link to the opsi manual.

Expert mode (3)
Activation of the check box activates the expert mode.

Expert mode

Main window (expert mode)
Is the expert mode activated (activation via the help Help) you can switch between the tile view of the standard mode and a list view. In the list view the products are displayed in a table among each other. The additional buttons Installation now and Reload are displayed (Figure 11).

Only if the installation as from now is not disabled the button Installation now is shown.
Figure 164. Kiosk (expert mode) - List view. (1) Check box to toggle view. (2) List view of the products. (3) Button for installing products (as from now). (4) Button for synchronisation with the server. (5) Detailed information of the product.

**Toggle view (1)**
Using the check box you can toggle between list view and the tiles view.

**List view (2)**
In the list view the products are displayed in a table among each other. In the column `ActionRequest` an action request can be set.

<table>
<thead>
<tr>
<th>Productid</th>
<th>ProductName</th>
<th>InstallationStatus</th>
<th>ActionRequest</th>
</tr>
</thead>
<tbody>
<tr>
<td>flashplayer</td>
<td>Adobe Flashplayer</td>
<td>installed</td>
<td></td>
</tr>
<tr>
<td>audacity</td>
<td>Audacity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>autot</td>
<td>autot=2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bcdein</td>
<td>BCDEin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdburnerxp</td>
<td>CD BurnerXP</td>
<td>installed</td>
<td></td>
</tr>
<tr>
<td>classic-shell</td>
<td>Classic Shell</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enpass</td>
<td>Enpass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>filezilla</td>
<td>FileZilla</td>
<td></td>
<td></td>
</tr>
<tr>
<td>firebird</td>
<td>Firebird SQL EDBMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>foto-aufmass</td>
<td>Foto-Aufmass Professional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freepdf</td>
<td>FreePDF Printer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ganttproject</td>
<td>GanttProject</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Flashplayer property description mms.cfg flash_player_admin_guide.pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advice</td>
<td>Uses empty template mms.cfg patched by non empty product-properties. Update.ins is only for security alignment.</td>
</tr>
<tr>
<td>Version at Server</td>
<td>32.0,0,270-1</td>
</tr>
<tr>
<td>Version at Client</td>
<td>None-None</td>
</tr>
</tbody>
</table>

Depending on product state the actions which can be set in the column `ActionRequest` may differ. Setting the action to `setup` yields to the installation or update of the product. The action `none` is for resetting an action which was set but was not executed until now.

**Installation now (3) - only available if the installation as from now was not disabled**
Clicking on the button *Installation now* opens a separate window showing the actions set. You can choose if the actions shall be executed now or later at the next standard event (Figure 12).
Figure 165. Kiosk (Expert mode) - Window for the execution of the actions set.

Window for the execution of the actions set

At this window (Figure 12) you can execute an installation as from now (button Installation now). In this case it would be advisable to close all other applications (i.e. to save your data) because the started installations may come in conflict with currently running applications. Using the button Install at next standard event (e.g. Reboot) closes the window and the actions will be proceed later at next standard event.

Admin mode

To execute the kiosk in the admin mode the config software-on-demand.admin-mode = true had to be set. In addition the kiosk must be executed as administrator.

Using the kiosk in admin mode allow to set product icons and screenshots for the products. Therefore you need to click with the right mouse button on the icon or screenshot or its respective placeholder. This opens a file dialog from which the desired image can be chosen. This works within the main window as well as in the detailed product view. A click on the button (Figure 13) opens a dialog to save the icons/screenshots on an opsi depot which means that these icons/screenshots are than also available for other kiosk installations.
After setting icons and screenshot within the kiosk the icons and screenshots are saved on the client only locally. To make these images available for kiosk installations on other clients they had to be copied to the ock-custom folder (e.g. with the kiosk as described above).

The config `software-on-demand.show-details` from opsi 4.0.7 has no effect on the kiosk anymore and can be deleted.

Logs of the opsi-client-kiosk:

The program logs to `C:\opsi.org\applog`.

9.17.5. Characteristics

The following apply to the software-on-demand module:

- Dependencies are resolved automatically
  - Software that depends on software from the Demand group, will automatically be set to setup (install), without interaction from the user.

opsi-client-kiosk: Customizable to Corporate Identity

The header bar of the main window (1) is customizable. Three files in the ock_custom directory (`/var/lib/opsi/depot/opsi-client-kiosk/files/app/ock_custom/skin`) are key:

- `header.png`
- `logo.png`
- `opsiclientkiosk.ini`

The `header.png` contains the image which is loaded to the left of the logo in this area, `logo.png` the logo.

The `opsiclientkiosk.ini` defines the text and its display, which will be shown in this area (Since opsi-client-kiosk version 4.1.1.7-2).
The applied desktop/startmenu icon can be changed by placing a `kiosk.ico` file (Windows) or `kiosk.png` file (Linux) in the `ock_custom/skin` directory (starting with opsi-client-kiosk version 4.1.1.7-2).

### 9.18. User Profile Management (free)

#### 9.18.1. Preconditions for the opsi extension User Profile Management

This module was a co-funded opsi extension and is released as a free module since 29 Apr 2015. Further details can be found in Section 9.1, “Activation of non-free modules”.

Technical preconditions are opsi 4.0.1 with the following package and product versions:

<table>
<thead>
<tr>
<th>opsi product</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-client-agent</td>
<td>&gt;=4.0.1-23</td>
</tr>
<tr>
<td>opsi-script</td>
<td>&gt;=4.11.2.1</td>
</tr>
<tr>
<td>python-opsi</td>
<td>&gt;=4.0.1.31-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opsi product</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>opsi-client-agent</td>
<td>&gt;=4.0.5.4-2</td>
</tr>
<tr>
<td>opsi-script</td>
<td>&gt;=4.11.4.17-1</td>
</tr>
</tbody>
</table>

This extension does not work in combination with the WAN extension! Please do not activate the login event on WAN clients.

#### 9.18.2. Introduction

The `opsi-script` has a number of special commands for making modifications in profiles. However, this works only on local profiles and is largely useless when using Roaming Profiles. With the opsi
extension User Profile Management it is now possible to make changes to these profiles. This is done by starting the opsi-script when the user logs in, in order to execute special userLoginScripts.

9.18.3. Concept

If the profiles cannot be patched while installing software on the machine, a clear distinction must be made between the machine part and the profile part of the installation. This can be done both within a script or by storing the profile part in a separate script. In many installations this is already done by executing the profile parts as part of a domain login script.

Depending on the practice, the profile parts of opsi products are available as part of the opsi scripts for installation and deinstallation, as well as part of a domain login script. The aim of this extension is to be able to integrate both variants as simple as possible in the new mechanism.

The core concepts of this opsi extension are:

- Executing special userLoginScripts when the user logs in
  As part of the user login, the opsi-script is started but executed in a special mode in which only userLoginScripts stored in the products are executed.

- Executing these scripts with administrative rights but in the context of the logged in user
  Domain login scripts are executed by the user with user rights. The opsi userLoginScripts are executed by the opsi-script, which runs with administrative rights. At the same time, the opsi-script moves into the context of the user who has logged in, so that the profile can be manipulated with the same commands as in a domain login script.

- Execution of these scripts within the opsi-service context
  The opsi userLoginScripts run within the opsi-service context and thus have information via script constants about the product name, version and package version currently being processed. Furthermore, the values of the product properties are available as well as any other information which may be queried via opsiservicalls.

Limitations:

- Even when using the opsi-WAN-extension, the userLoginScripts are not loaded from the local cache, but are retrieved from the server.

9.18.4. New and changed opsi-script functions

- Call parameter /allloginscripts or /loginscripts
  Executing opsi-script in the opsi-service context with the additional parameter /allloginscripts or /loginscripts essentially has the following effects:
  - The products that have a userLoginScript are determined.
    With /allloginscripts the userLoginScripts are executed for all of these products.
    With /loginscripts, only the loginscripts are executed on a client for which the client knows the corresponding product, i.e. the product is installed or was installed.
  See also: Section 9.18.6, “Configuration”
The logged in user will be identified, and constants for the current user, e.g. `%CurrentAppdataDir%` will be directed to the corresponding directories of the logged in user. Likewise, registry operations (Registry sections and GetRegistryString) which refer to HKCU are carried out in such a way that the data comes from the registry branch of the user.

- **Call parameter `/silent`**: The call parameter `/silent` ensures that the window of opsi-script is not displayed during script processing.

- **Function `GetScriptMode`**: In order to distinguish in a script in which mode the script is currently being executed, the function `GetScriptMode` returns two possible values:
  - **Machine**: The script is not executed as `userLoginScript` (but e.g. as setup or uninstall script).
  - **Login**: The script is executed as `userLoginScript`.

- **New primary section `ProfileActions`**: This new section can be used to bundle actions for user profiles. Here a syntax can be used which enables this section to be used both as part of a normal login script and as a `userLoginScript`. This primary section is evaluated in different ways, depending on whether the script is running in machine mode or login mode (i.e. as userLoginScript).
  - **Login**: If a script runs as `userLoginScript` and contains a section `ProfileActions`, script processing is started in this section (and not in `Actions`).
  - **Machine**: If a script runs as a normal installation script, section `ProfileActions` can be called as a sub-section, similar to a `sub`-section. The following applies to the processing of this section: `/AllNtUserDats` is implicitly set for all registry-section calls. `/AllNtUserProfiles` is implicitly set for all `Files`-section calls. Since version 4.11.3.2 the following also applies: `/AllNtUserProfiles` is implicitly set for all `Patches`-section calls.

- **Registry sections**:
  - Registry sections that work on HKCU or HKEY_CURRENT_USER are executed in login script mode so that the changes end up in the branch of the logged-in user. The same applies to the functions `GetRegistryStringValue*`.
  - Registry sections which are called in normal mode (Machine) with the modifier `/AllNtUserDats` can now contain the root HKCU or HKEY_CURRENT_USER in the `openkey` command. This makes it possible to run the same registry section in different modes.

- **Winbatch sections with `/RunAsLoggedOnUser`**: Even if opsi-winst is started via the login event, it runs in the SYSTEM context and not in the context of the user who has just logged in. A winbatch section with the option `/RunAsLoggedOnUser` can be used to start a process in the context of this user.

- **Avoid unnecessary script execution:**
The command `saveVersionToProfile` can be used to store in the current profile that the userLoginScript for this product has been executed in this version. The string function `readVersionFromProfile` or the boolean function `scriptWasExecutedBefore` can be used to check whether the userLoginScript for this product has already run in this version and whether it needs to be executed again. For this purpose, this function first reads in a possibly existing version stamp from the profile (as is possible with `readVersionFromProfile`) and compares it with the currently running version. The return value (true/false) results from the comparison. Then the current values are written back to the profile (as is possible with `saveVersionToProfile`). So you only need this `scriptWasExecutedBefore` function in an `if`-statement to check whether the script has already run.

Furthermore, the string list function `getProductMap` provides an info map from which you can see whether the current product is installed or uninstalled etc.

- Any kind of `ExitWindows` command will be ignored.

Logging

The logs of userLoginScripts are written to:
```
c:\opsi.org\log\<user login name>_login.log
```
These log files are also transferred to the opsi server. A new log file is appended to an existing one. The opsi-server ensures that these files are limited in size (max. 5 MB). On the opsi server, these logs are stored as `/var/log/opsi/userlogin/<clientid>.log`

In the opsi management interface (opsi-configed), these logs are displayed in an additional sub-tab `userlogin` in the `Log files` tab.

### 9.18.5. Examples of userLoginScripts

First, two examples that are structured as they could also be used in domain login scripts.

A very simple generic example:

```
[Actions]
requiredWinstVersion >= "4.11.3.2"
Message "Example Profile Patch ...."

Files_profile_copy
Registry_currentuser_set
Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini"

[Files_profile_copy]
copy "%Scriptpath%\profiles\*.*" "%CurrentAppdataDir%\ACME"

[Registry_currentuser_set]
openkey [HKCU\Software\ACME]
set "show_greeting_window" = "no"

[Patches_profile_ini]
add [secdummy] dummy1=add1
```
An example for Firefox configuration:

```
[Actions]
requiredWinstVersion >= "4.11.3.2"
Message "Firefox Profile Patch ...."

DefVar $akt_profile_ini$
DefVar $rel_prefs_path$

comment "check for existing profile ...
Set $akt_profile_ini$ = "%CurrentAppdataDir%\Mozilla\Firefox\profiles.ini"
if FileExists($akt_profile_ini$)
    Set $rel_prefs_path$ = GetValueFromInifile($akt_profile_ini$,"Profile0","Path",""
    if FileExists("%CurrentAppdataDir%\Mozilla\Firefox\"+$rel_prefs_path$)
        comment "We found the profile and will now patch it ...."
    endif
else
    comment "no firefox profile found for user"
endif
```

Next we show an example, which extends the first one, that shows how to also delete items from the profile. Depending on whether the product is installed or was uninstalled on the computer, a different part of the script is executed:
Now an example that uses the setup script (setup32.ins and delsub32.ins) to avoid unnecessary duplication of the code:

setup32.ins:

[Actions]
requiredWinstVersion >= "4.11.3.2"

DefVar $MsiId$
DefVar $UninstallProgram$
DefVar $ProductId$
DefVar $InstallDir$
Set $ProductId$ = "ACME"
Set $InstallDir$ = "%ProgramFiles32Dir%\ACME"

if GetScriptMode = "Machine"
    comment "Show product picture"
    ShowBitmap "%ScriptPath%\" + $ProductId$ + ".png" $ProductId$
endif

if FileExists("%ScriptPath%\delsub32.ins")
    comment "Start uninstall sub section"
    Sub "%ScriptPath%\delsub32.ins"
endif

Message "Installing " + $ProductId$ + " ..."

comment "Start setup program"
Winbatch_install

    comment "Patch the local Profiles ..."
    Registry_currentuser_set /AllNtUserDats
    Files_profile_copy /AllNtUserProfiles
    Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini" /AllNtUserProfiles
endif

if GetScriptMode = "Login"
    comment "login part"
    Files_profile_copy
    Registry_currentuser_set
    Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini"
endif

[Winbatch_install]
"%ScriptPath%\setup.exe" /sp- /silent /norestart

[Files_profile_copy]
copy "%Scriptpath%\profiles\*.*" "%CurrentProfileDir%\Appdata\ACME"

[Registry_currentuser_set]
openkey [HKCU\Software\ACME]
set "show_greeting_window" = "no"

[Patches_profile_ini]
add [secdummy] dummy1=add1

delsub32.ins:
Message "Uninstalling " + $ProductId$ + " ...

if GetScriptMode = "Machine"
  comment "The machine part ...
  Set $UninstallProgram$ = $InstallDir$ + "\uninstall.exe"
  if FileExists($UninstallProgram$)
    comment "Uninstall program found, starting uninstall"
    Winbatch_uninstall
  endif
  ; does also work since 4.11.2.1
  Registry_currentuser_del /AllNtUserDats
  Files_profile_del /AllNtUserProfiles
endif

if GetScriptMode = "Login"
  comment "The profile part ...
  Files_profile_del
  Registry_currentuser_del
endif

[Winbatch_uninstall]
"$UninstallProgram$" /silent /norestart

[Files_profile_del]
del -s -f "%CurrentAppdataDir%\ACME"
del "%userprofiledir%\opsi-winst-test.ini"

[Registry_currentuser_del]
deletekey [HKCU\Software\ACME]

Now an example which is a variant of the previous example. The code is simplified by using the new primary section ProfileActions and the script can be used both as an installation script and as a userLoginScript.
Now a variant, which notes in the profile whether the script for this product has already been executed in this version and for this user. A line with the product information is written to the file %AppData%\opsi.org\userLoginScripts.ini.
In order to use the User Profile Management extension, the login event must be activated in the configuration of opsiclientd. For this event (if the current opsi-client-agent is installed on the client) opsi-script will be started with the additional parameter /allloginscripts or /loginscripts.

- /allloginscripts means that when you log in all loginscripts that are known to the server are executed, regardless of whether the corresponding product is known to the client (i.e. the product is installed or was installed) or not. This is the default.

- /loginscripts means that when you log in, only those loginscripts are run on a client for which the client knows the corresponding product, i.e. the product is installed or was installed. (Technically: where there a productOnClient object for this client exists with:

  - (installationStatus= installed)
  - or actionResult = successful) and (lastAction = uninstall).

Login scripts for products that were never installed on this client will not be executed.

You can activate the login event on the command line as follows: (usually you only want to activate individual clients for testing, therefore the value is false on the server side)

```bash
opsi-admin -d method config_createBool opsiclientd.event_user_login.active "user_login active" false
```
The parameter `/silent` can also be used as an additional `opsi-script` parameter, which prevents the `opsi-script` window from being displayed.

```
opsi-admin -d method config_createUnicode
opsiclientd.event_user_login.action_processor_command "user_login action_processor"
"%action_processor.command% /sessionid %service_session% /loginscripts /silent"
"%action_processor.command% /sessionid %service_session% /loginscripts /silent"
```

Settings configured this way can be modified in the opsi management interface in the `Host parameters` tab, server or client-specific.

### 9.18.7. Notification

If you have activated the login event (as described above), you will see the `user_login_notifier` after each login:

![User Login Notifier](image)

*Figure 167. User Login Notifier*

### 9.19. opsi Installation on Shutdown (free)

#### 9.19.1. Introduction

By default, the installation of opsi packages is starts when the client boots. The user then has to wait for the software installation to finish, before login is possible. It can therefore be desirable to do most of the software installations when the client is shutting down.

The opsi module for installation during shutdown provides exactly this functionality. Installation on shutdown can be activated individually for certain clients.

#### 9.19.2. Preconditions for Installation on Shutdown

The opsi module Installation on Shutdown can be used on clients with Windows XP or above. The required components are part of the opsi `opsi-client-agent` packages.

The package `opsi-client-agent` must be version 4.0.2.3-2 or above with `opsiclientd` version 4.0.75 or above.

The cofunding of this project is completed, and thus the module installation on shutdown was released with opsi version 4.0.5.4-2 and is now part of the free opsi system. A corresponding
modules file is therefore no longer required. For older \textit{opsi-client-agent} packages, i.e. 4.0.2.3-2 to 4.0.5.4-2, a corresponding modules file is required. Further details on cofunding projects can be found in Section 9.1, “Activation of non-free modules” and at co-financed opsi extensions.

Some technical restrictions and constraints exist in the following cases:

- WAN extension: the Installation on Shutdown module is currently not available for clients that also use the WAN extension. Under some conditions, running the WAN extension on a client requires the use the local configuration files, which interfere with the state handling of the installation on shutdown mechanism.

- Group Policies: part of the mechanism for Installation on Shutdown is based on shutdown scripts per Local Group Policy. Other Group Policies might override these local configurations. In this case, the required configuration for running these shutdown scripts should be included in higher-level group policies. See Section 9.19.4.2, “Installing by shutdown script” for the required configuration.

- If the install on shutdown has been triggered by a shutdown or a reboot, the Windows API does not allow to convert a shutdown into a reboot (or vice versa). In the case of a shutdown, products that require a reboot will only be fully installed the next time the computer is started.

- Windows Home Edition: Windows Home does not provide the required Local Group Policy shutdown script mechanism. Therefore, Installation on Shutdown cannot be used on Windows Home Edition.

- Windows 2000: On Windows 2000 the maximum waiting time for shutdown scripts is 10 minutes, after which the installation is automatically canceled by Windows. For this reason, the module cannot be used on Windows 2000 clients.

\textbf{9.19.3. Activating Installation on Shutdown}

The required components for Installation on Shutdown are part of the current \textit{opsi-client-agent} package. Since opsi version 4.0.5.4-2 the no corresponding modules file is required any more.

For suitable clients (see Section 9.19.2, “Preconditions for Installation on Shutdown”) simply switch the product property \texttt{on\_shutdown\_install} of the package \textit{opsi-client-agent} to \texttt{on} and set the action request for \texttt{opsi-client-agent} to update (or setup).

Further settings are normally not required.

If group policies are already being used on the clients, there might be a conflict with the local opsi client group policy setting. In case of such a conflict, the opsi shutdown script group policy entries of the opsi-client-agent package should be disabled (\texttt{on\_shutdown\_install\_set\_policy = off}) and should be integrated manually into your own group policy management, see Section 9.19.4.2, “Installing by shutdown script”.

The installation at shutdown is carried out in addition to the installation at startup. Usually this is the best way to ensure that the clients always have the current security updates installed, even if the client was off for a long time (when the user was on holiday for instance). If required, the standard
Installation on Startup can be disabled, see Section 9.19.4.5, “Special Configuration of Installation on Shutdown”. Any installations that have already started will be continued when the computer is started, see Section 9.19.4.4, “Configuration of opsiclientd”.

- Windows does not distinguish between shutdown and reboot. The installation during shutdown is carried out both during shutdown and reboot and it is not possible to distinguish between these two cases when executing a script. Also, the Windows API does not allow a shutdown to be converted into a reboot (or vice versa), nor does it allow the shutdown/reboot to be canceled. If individual software packages require a multi-phase installation with reboot, the installation will only continue the next time the client is started.

### 9.19.4. Technical Concept

The following explanations serve for a better understanding of the technical relationships for special configuration variants, as well as for investigation in the event of an error. Normally all necessary settings are made by the `opsi-client-agent` package.

#### Overview

The installation on shutdown module is based on the interaction of various system components. An essential component is the use of the Windows shutdown script mechanism via local group policy. Shutdown scripts enable tasks to be carried out when the user is already logged off and all user tasks have finished, but all system services are still running.

The shutdown script performs an opsi task, which triggers the opsi system service `opsiclientd` to start the installation process and waits for it to be completed. The system waits for the installation process to finish and then shuts down. The system makes no distinction between a shutdown and a reboot, so that the installation is also triggered with a reboot.

The opsi client service `opsiclientd` is configured to process the action `on_shutdown`, which starts the installation process. If reboots are required for the installation, the precondition `installation_pending` is responsible for the control of the process. If a reboot is required during the installation during shutdown, the precondition `installation_pending` (regardless of whether `gui_startup` is activated or not) continues the installation the next time the system is started, possibly also with further reboots. In the `installation_pending` state, if further reboots are required, no installation is carried out during shutdown. Otherwise there would be no reboot between the installation during startup and the installation during shutdown. I.e. the current installation continues when the system is booted, but not when the system is shut down, as otherwise there would be no reboot between the individual installation phases.

The two components are described in detail below.

#### Installing by shutdown script

When the system is shut down, a shutdown script that triggers the installation is executed via the corresponding registry entries per Local Group Policy. The registry entries correspond to the settings that can also be created with the group policy editor `gpedit.msc`. 
This is how you can use the Group Policy Editor to create a shutdown script entry:

- Local Computer Policy
- Computer Configuration
- Windows Settings
- Scripts (Startup/Shutdown)
- Shutdown
- Scripts - Add - Browse
- C:\Program Files\opsi.org\opsi-client-agent\on_shutdown\doinstall32.cmd (or doinstall64.cmd for 64Bit systems)

The waiting time for shutdown scripts is set to infinite (0 seconds) so that the system waits until the installation is complete:

- Local Computer Policy
- Computer Configuration
- Administrative Templates
- System - Scripts
- Maximum wait time for Group Policy scripts
- Setting - Enabled - Seconds: 0

The shutdown script doinstall32.cmd or doinstall64.cmd changes the working directory and triggers the on_shutdown event:

```bash
echo Start opsi product installation ...
cd "%ProgramFiles%\opsi.org\opsi-client-agent\on_shutdown"
opsiclientd_event_starter.exe --event=on_shutdown
```

or for 64-bit systems:

```bash
echo Start opsi product installation ...
cd "%ProgramFiles(x86)%\opsi.org\opsi-client-agent\on_shutdown"
opsiclientd_event_starter.exe --event=on_shutdown
```

The opsiclientd_event_starter waits for the installation to finish, so the system shutdown is delayed until then.

**Registry Entries for executing the shutdown script**

These registry entries are set by the opsi-client-agent package and lead to the execution of the specified shutdown script on WinXP / 32Bit. For 64-bit systems the script name is doinstall64.cmd
(instead of `doinstall32.cmd`).

These are the registry entries for Windows 64-bit (Vista / Win7 / Win8 / Win10):

```plaintext
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\State\Machine\Scripts\Shutdown\0]
 "GPO-ID"="LocalGPO"
 "SOM-ID"="Local"
 "FileSysPath"="C:\\WINDOWS\\System32\\GroupPolicy\\Machine"
 "DisplayName"="opsi shutdown install policy"
 "GPOName"="opsi shutdown install policy"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\State\Machine\Scripts\Shutdown\0\0]
 "Script"="C:\\Programme\\opsi.org\\opsi-client-agent\\on_shutdown\\doinstall32.cmd"
 "Parameters"=""
 "ExecTime"=hex(b):00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\System\Scripts\Shutdown\0]
 "GPO-ID"="LocalGPO"
 "SOM-ID"="Local"
 "FileSysPath"="C:\\WINDOWS\\System32\\GroupPolicy\\Machine"
 "DisplayName"="opsi shutdown install policy"
 "GPOName"="opsi shutdown install policy"

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\System\Scripts\Shutdown\0\0]
 "Script"="C:\\Programme\\opsi.org\\opsi-client-agent\\on_shutdown\\doinstall32.cmd"
 "Parameters"=""
 "ExecTime"=hex:00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system]
 "MaxGPOScriptWait"=dword:00000000
```
Configuration of opsiclientd

The opsi client service `opsiclientd` has additional entries in the configuration file `opsiclientd.conf` for the new event `on_shutdown`. Here are all relevant entries:
The precondition `installation_pending` indicates whether an installation is still in progress. If the status `installation_pending` is still set to true at the end of the script, this means that the current installation requires a reboot and has not yet been completed. In normal operation without installation at shutdown, the sections for the new action `event_on_shutdown` are deactivated.

For a client with activated installation on shutdown this is the required configuration:

```plaintext
[event_gui_startup]
active = True

[event_gui_startup{installation_pending}]
active = True

[event_on_shutdown]
active = False

[event_on_shutdown{installation_pending}]
active = False

[precondition_installation_pending]
installation_pending = true
```

So the only difference here is:

```plaintext
[event_on_shutdown]
active = True
```

This setting is controlled via the property switch `on_shutdown_install` of the `opsi-client-agent` package.
The precondition `precondition_installation_pending = true` means that the installation process has not completed yet. This state persists through one or more reboots until the installation is completed. If the installation requires a reboot, the installation will be continued when the system is started with the configuration `[event_gui_startup {installation_pending}] active = True`. This setting must not be changed, since an installation that has already started must be completed before the user can log on.

The entry `[event_on_shutdown{installation_pending}] active = False` must always remain on False, since if an installation has started there would otherwise be no reboot between the installation phases during startup and shutdown.

As soon as the installation process has completed, the precondition is set to `installation_pending = false`, so the Installation on Shutdown becomes active again.

**Special Configuration of Installation on Shutdown**

Normally nothing more is required than an up-to-date `opsi-client-agent` package to start the installation at shutdown, as described in Section 9.19.3, “Activating Installation on Shutdown”. For suitable clients, the `opsi-client-agent` product property `on_shutdown_install` can be used to activate the installation during shutdown.

By default, the installation at startup also remains active. This ensures that a client that has been switched off for a long time (e.g. during the user’s vacation) always receives the latest software, before the user can log in.

If this is not desired, the installation can be deactivated at startup. Since the configuration of the `opsi-client-agent` can also be done centrally via the web service (see: Section 6.1.3.12, “Configuration via web service (Host Parameter)”), we recommend that the following ’host parameter ‘is created:

- `opsiclientd.event_gui_startup.active` (boolean, default: `true`)

By using this `host parameter` the gui_startup event can then be activated or deactivated for a specific client. The `host parameter` can be created via `opsi-configed` or `opsi-admin`.

To create the `host parameter` via `opsi-admin`, execute the following command on the `opsi-config-server`:

```
opsi-admin -d method config_createBool opsiclientd.event_gui_startup.active "gui_startup active" true
```

The default value `true` corresponds to the value in the supplied `opsiclientd.conf`.

If the installation on startup is to be deactivated for an install_on_shutdown client, the corresponding `host parameter` is configured as follows:

- `opsiclientd.event_gui_startup.active: false`

However, this should only be done in exceptional cases. The settings with the additional condition `installation_pending` should not be changed under any circumstances. The default values are required
to ensure correct control of the process.

To set the host parameter via opsi-admin, the following command must be executed on the opsi-config-server (in this example for a client with the opsi-host-Id myclient.domain.de):

```
opsi-admin -d method configState_create opsiclientd.event_gui_startu.active
myclient.domain.de false
```

This configuration means that when the computer is started, no connection is established to the opsi-config-server and thus no installation takes place. Except when an installation has started, which is indicated by the additional condition installation_pending. In this case the setting event_gui_startup {installation_pending} will make the installation continue at system start. If another reboot is required, the setting event_on_shutdown {installation_pending} (which must also not be changed) prevents the installation from being continued even after shutdown. Otherwise there would be no system restart between the individual installation phases.

**Local Logfiles**

During Install on Shutdown two logfiles are written:

- C:\opsi.org\log\doinstall.log
- C:\opsi.org\log\opsiclientd_event_starter.log

These usually have the following content:

**doinstall.log:**

```
doinstall32.cmd started
Current date: 29.01.2013
```

**opsiclientd_event_starter.log:**

```
[1] [Okt 06 18:49:44:435] opsiclientd_shutdown_starter: version: 4.0.7.0
[5] [Okt 06 18:49:44:450] host_key=***confidential***
[6] [Okt 06 18:49:45:107] JSON Bench for backend_info "params":[],"id":1} Start:
18:49:44:450 Time: 00:00:00:657
isInstallationPending
```
[6] [Okt 06 18:49:45:529] JSON Bench for isInstallationPending "params":[],"id":1}
Start: 18:49:45:232 Time: 00:00:00:297
[5] [Okt 06 18:49:45:622] resultstring={"id": 1, "result": false, "error":null}
[5] [Okt 06 18:49:45:622] No installation pending - fine
fireEvent
[6] [Okt 06 18:49:45:966] JSON Bench for fireEvent "params": ["on_shutdown"],"id":1}
Start: 18:49:45:622 Time: 00:00:00:344
isEventRunning
[6] [Okt 06 18:49:56:935] JSON Bench for isEventRunning "params": ["on_shutdown"],"id":1}
Start: 18:49:56:467 Time: 00:00:00:468
[5] [Okt 06 18:50:02:076] calling: isEventRunning,[on_shutdown]
isEventRunning
[6] [Okt 06 18:50:07:686] JSON Bench for isEventRunning "params": ["on_shutdown"],"id":1}
Start: 18:50:07:686 Time: 00:00:00:500
isEventRunning
[6] [Okt 06 18:50:19:030] JSON Bench for isEventRunning "params": ["on_shutdown"],"id":1}
Start: 18:50:18:765 Time: 00:00:00:265
[5] [Okt 06 18:50:19:171] calling: isEventRunning,[on_shutdown{user_logged_in}]
isEventRunning
9.20. opsi Feature SilentInstall (free)

The opsi feature SilentInstall—provides for administrators the ability to install software on a client without the logged on user being disturbed. This chapter describes the characteristics of this feature and offers a guideline for configuring this new installation method.

9.20.1. Preconditions for the Silent Installation

For using this feature, opsi version 4.0.3. or above is required. Basically the opsi-client-agent version 4.0.3.1 or above is required.

9.20.2. Overview of the SilentInstall feature

The SilentInstall method offers the ability to install a pre-defined list of products on a client, without the user having to interrupt his work. Unlike the installation by the onDemand-Event (push Installation), the SilentInstall method does not display anything on the user desktop.

These log files are rewritten each time and can be checked in case of an error.
All displays are suppressed and are not to be seen on any desktop. Of course this feature bears some risk. In case of a problem, e.g. a syntax error of the \textit{opsi-winst} script, there is no way to interact with the installation process, for no dialog windows are shown. So this would result in the \textit{opsi-winst} and so the Event processing not coming to an end, and so no more events will be executed. To avoid this "Worst case scenario", the maximum installation time is limited by a timeout configuration. This timeout value might have to be adapted in case of an extended installation. For further information, see the configuration chapter.

Another very important ability of this feature is the predefined list of products to be installed silently. Contact to the service is established, but different from the usual procedure, the ActionRequests given by the service are ignored. The list of software to be installed is defined by an \textit{opsi-client-agent} configuration. The "setup" action will be executed for all the products on this list and they do not have to be set to setup. As usual after processing the setup script, also the update script will be executed, if there is one. \textbf{No product dependencies will be resolved.} So either no products containing product dependencies should be installed by the SilentInstall feature, or all the products from the dependency list must be added to the SilentInstall product list. As usual the installation process is completed by sending the installation status and installation logfiles to the service. In summary it is recommended to use the SilentInstall only for products, that meet the following requirements:

- small packets or installations only
- little system load: some software installations, so for instance most of the MSI installations, request during installation most of the clients ressources. This could result in a poor system performance remaining for the user.
- installable within a fixed amount of time: the default timeout for this event is set to 300 seconds. If the installation process has not completed within the timeout, the 'opsi-winst'process will be terminated and so the event can be completed.
- no reboots required: software requesting a reboot should not be installed from the SilentInstall. With the default configuration the event is configured not to process any reboot requests. Without this safety configuration the \textit{opsi-client-agent} could reboot the client without any warning to the user. This could result in loss of data if there is a logged on user. This could result in an inoperable software installed by SilentInstall without reboot.

Within the default configuration swaudit and hwaudit are installed by this method. The inventory products of opsi meet all the requirements above and so are applicable for this method. With the default configuration the opsi hard- and software inventory are generated on demand, without the need to set the setup action request with the \textit{opsi-configed}. With this method the inventory information can be generated in real-time operation. Also applicable would be any configuration products, that perform automatic repairs or restore client patches.

\section*{9.20.3. Executing the Silent Installation}

This event will not be triggered automatically like other events. So there are two ways to perform this event.

The first way is to trigger the event from the opsi webservice, like e.g.:
opsi-admin -d method hostControl_fireEvent silent_install client.domain.local

So this command is scriptable and can be used within scripts that can be combined with an *at-job* to plan the execution of the event.

As an alternative the event can be triggered by a timer event after a certain amount of time. The default configuration for this event is 6 hours. This value presumes, that a work station usually is in use for 8 hours. So the event would be executed once a day after six hours of uptime. For more information on configuring and activating this event see the following configuration chapter.

### 9.20.4. Configuring the opsi-feature: SilentInstall

This chapter is about the default configuration of this feature. The default *opsiclientd.conf* has got a SilentInstall event. This listing just shows the important options:

**Standard Event SilentInstall:**

```
[event_silent_install]
process_shutdown_requests = false
action_processor_productIds = swaudit,hwaudit
action_processor_command = %action_processor.command% /productlist
%action_processor_productIds% /silent
action_processor_desktop = winlogon
action_processor_timeout = 300
```

- **action_processor_productIds**
  - This option is an important new property for the event control. For all events that perform product actions, this option can define a list of products. The product list must be given as a comma separated list.
- **process_shutdown_request = false**
  - This configuration suppresses reboot requests from the *opsi-winst*.
- **action_processor_command**
  - This prepares the call of *opsi-winst*.
- **action_processor_desktop**
  - This option defines the desktop to display the *opsi-winst* GUI.
- **action_processor_timeout**
  - This option sets the timeout for terminating the *opsi-winst*-process.

The second event is the Timer Event, which triggers the event after a certain amount of time:

**Default Timer Event for SilentInstall**
• super
  ○ This option defines the event to inherit properties from. As the default configuration the
  Timer-Event inherits from the event silent_install.
• type
  ○ This option defines this event configuration to be a Timer-Event.
• active
  ○ as default this event is not active. To activate it, set this option to true.
• interval
  ○ This option defines the interval to fire the event. The default value is set to 6 hours, so after
  six hours of uptime the event is triggered the first time and then every other six hours. So this
  interval should (like any timer interval) not be too short, otherwise the event would be
  performed most of the time and thereby block the execution of other actions. On the other hand
  the interval also should not be too long, for the opsi-client-agent must be running all that time
  until the event is triggered. If the client or the opsi-client-agent always is restarted before the
  interval elapsed, this event never will be triggered.

Also the SilentInstall event could be triggered by another system event detected by an WMI request.
Therefore a wql option can be specified. How to specify a wql option is to be seen in the
event_net_connection section. If the wql option is used, the event should be set to active = false as
default, so it can be activated later on when requested.

To trigger the event by a timer, usually it only needs to set a host parameter. Therefore at first a
default configuration has to be created. In this case it is sufficient to activate the Timer Event.

To create the standard option the following host parameter are to be created by the opsi-admin. Also
this configuration could be created by the opsi-configed:

opsi-admin -d method config_createBool opsiclientd.event_timer_silentinstall.active
"event_timer_silentinstall active" false

So at first this event is disabled for all the clients. Then the event can be enabled for single clients:

opsi-admin -d method configState_create opsiclientd.event_timer_silentinstall.active
silentclient.domain.de true
To define the products to be installed, the following entry must be set. If for instance instead of *swaudit* and *hwaudit* the product firefox shall be installed, the entries should be created as described above:

```
opsi-admin -d method config_createUnicode
opsiclientd.event_silent_install.action_processor_productIds "event_silent_install productIds" "swaudit,hwaudit" "swaudit,hwaudit"
```

With this option as the default for all clients the product list for the Silent Install Event is set to swaudit and hwaudit. To change the product list for a single client into firefox execute the following command:

```
opsi-admin -d method configState_create
opsiclientd.event_silent_install.action_processor_productIds client.domain.de "firefox"
```

As you can see, the product list can be different for each client.

### 9.21. opsi Setup Detector (free)

#### 9.21.1. opsi Setup Detector Quickstart

After the first start of the opsi-setup-detector a configuration mask appears. The following information is required:

- **fullname**: (Used for entries in the changelog.txt)
- **email_address**: (Used for entries in the changelog.txt)
- **workbench_path**: Path to the directory in which the opsi packages are going to be created. This is ideally the path to the place where the *opsi_workbench* of your opsi-server is mounted.
Figure 168. opsi-setup-detector Needed configurations at the first start

At the start page select the desire task and follow the dialogs (click Next step)
9.21.2. Introduction: opsi Setup Detector (opsi-setup-detector) What is this?

The creation of opsi-packages from available setup programs is a recurring task when operating opsi. The opsi-setup-detector should help you to simplify this in two ways:

- Detection of the type of the setup program and the determination of the necessary command line parameters
- Creation of an opsi package with the necessary scripts for installation and deinstallation.
- If necessary, include the opsi-package-builder for further processing of the provided package or to build and install the package.

Most setup programs are not written by the developers themselves, but are based on different frameworks. The opsi-setup-detector now tries to find patterns in the setup program which are specific for certain frameworks and to recognize the framework on the basis of these and thus to determine the necessary information.

This can lead to you getting a finished opsi package with just a few clicks. Often you get an opsi-package to which you have to add further information after the first installation of the program (e.g. with the created opsi-package).

Of course there can also be problems. The most common are described in the chapter Opşı-setup-detector: Problems and solutions Section 9.21.9, “Opsi-setup-detector: Problems and solutions” There is also described how you can help us to improve the opsi-setup-detector after solving a problem.

9.21.3. Installation of the opsi-setup-detector and opsi PackageBuilder
Installation des opsi PackageBuilder

The opsi PackageBuilder is currently available for Windows and Linux and MacOS.

The installation files / packages of the opsi-package-builder can be found here: https://forum.opsi.org/viewtopic.php?p=32473#p32473

Dort findet sich im oberen Teil die Links auf die Installationspakete für Windows, Linux und MacOS.

The {opsi-package-builder} is not made by uib but by Holger Pandel from the opsi-community (thank you!)

Der {opsi-package-builder} unterliegt einer OpenSource Lizenz:
https://github.com/pandel/opsiPackageBuilder/blob/master/LICENSE_DE

Der {opsi-package-builder} hat eine eigene Dokumentation welche mit installiert wird.

Installation of the opsi-setup-detector

The opsi-setup-detector is currently available for Windows and Linux.

You can install the opsi-setup-detector via opsi:

The package opsi-setup-detector belongs to the opsi standard products and should be installed on your opsi-server. If not, use:

```bash
opsi-package-updater install opsi-setup-detector
```

to install it on the opsi-server.

A setup program to install the opsi-setup-detector without opsi can be found at:
https://download.uib.de/opsi4.1/misc/helper/

The functionality of the Linux version is limited, since certain programs for analyzing setup programs from certain frameworks are not available for Linux:

- Detailed analysis of Inno-Setups due to the lack of innounpack.exe for Linux
- Detailed analysis of wix-setups due to the lack of dark.exe for Linux

The opsi product opsi-setup-detector has a dependency on the opsi product opsi-package-builder. The opsi-setup-detector uses the opsi-package-builder if available, but can for the most part also work without it. The installation of the opsi-package-builder is recommended.

9.21.4. Requirements in order to use the opsi-setup-detector

Recommendation:

- You should have mounted the share opsi_workbench of your opsi-server on your computer.
- Ideally you should have the opsi-Package-Builder installed on your computer.
9.21.5. Integration of the opsi-package-builder through the opsi-setup-detector

<table>
<thead>
<tr>
<th>Create Mode</th>
<th>Build Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>☑ create opsi product files</td>
<td>☑ build</td>
</tr>
<tr>
<td>☐ create opsi product files and build package</td>
<td>☐ build and install</td>
</tr>
<tr>
<td>☐ create opsi product files and start interactive PackageBuilder</td>
<td></td>
</tr>
</tbody>
</table>

Figure 170. opsi-setup-detector Create and Build Modes

On the tab Create you will find a selection for the creation mode. The default is Create opsi product files.

If you choose one of the other modes, an installed opsi-package-builder will be called.

In case of ... and build opsi package the opb is called in the background to build the directory created by the opsi-setup-detector with the opsi files to a package. The checkboxes Build mode can be used to select whether the build should be silent, if the package should be built and if the built package should be installed.

Without any of these options, this call corresponds to the last build mode: ... and start interactive package builder.

9.21.6. Usage of the opsi setup detector

Opsi-setup-detector Start and necessary configurations

The opsi-setup-detector can be started from the programs menu and can be found there under opsi.org. Under Windows the opsi-setup-detector is also integrated into the context menu of the explorer in order to call the setup program directly for analysis with the right mouse button.
Figure 171. opsi-setup-detector Necessary configuration on first startup

After the first start of the opsi-setup-detector a configuration mask appears. The following information is required here:

- **fullname**: (Used for entries in the changelog.txt)
- **email_address**: (Used for entries in the changelog.txt)
- **workbench_path**: Path to the directory in which the opsi packages are going to be created. This is ideally the path to the place where the `opsi_workbench` of your opsi-server is mounted.

After all needed configurations has be done and saved, you will see the startpage.
On the main window, select the desired task and follow the dialogs or select the Next step button.

The offered tasks are grouped by:

- Windows
- Linux
- MacOS
- multi platform

The offered tasks for Windows:

1. **Analyze single setup file and create an opsi package**
   Here, a setup file is analyzed and the entire process is run until an opsi package is created. This process is described in the next chapter.

2. **Analyze two setup files (32 and 64 bit) and create an opsi package**
   The procedure is the same as in point 1 above with the following differences:
   Two setup programs for the 32 and 64 bit architectures are queried and analyzed. The product gets an additional property: install_architecture with the possible values: 32bitonly, 64bitonly, both, systemspecific.

3. **Analyze single setup file**
   runs, similarly to point 1 above, only that is aborted after the analysis of the setup program.

4. **Create an opsi package template**
   This point does not ask for a setup file, but creates a template similar to the opsi product opsi-template only that here the information from the product configuration is already taken over.

Die Angebotenen Aufgaben für Linux:
1. Analysiere einzelne Linux Installer-Datei und erzeuge ein opsi Paket
   Hier wird von einer Installer-Datei ausgegangen und der gesamte Ablauf bis zur Erzeugung eines
   opsi-Paketes durchlaufen. Dieser Prozess ist im nächsten Kapitel beschrieben.

2. Eine opsi Paketvorlage (Template) für Linux erzeugen
   Dieser Punkt fragt nicht nach einer Installer-Datei, sondern erstellt ein Template analog dem opsi-
   Produkt opsi-template nur das hier die Angaben aus der Produktkonfiguration bereits
   übernommen werden.

The offered tasks for MacOS:

1. Analyze single MacOS Installer file and generate an opsi package
   Here it starts from a MacOS installer file and then goes through the whole process to create an
   opsi package. This process is described in the next chapter.

2. Create an opsi package template for MacOS
   This point does not ask for a setup file, but creates a template comparable to the opsi product
   opsi-template with the exception that here the specifications from the product configuration are
   already taken over.

Die Angebotenen Aufgaben für MacOS:

1. Analysiere einzelne Setup Datei und erzeuge ein opsi Paket
   Hier wird von einer Setup-Datei ausgegangen und der gesamte Ablauf bis zur Erzeugung eines
   opsi-Paketes durchlaufen. Dieser Prozess ist im nächsten Kapitel beschrieben.

2. Analysiere zwei Setup Dateien (32 und 64 Bit) und erzeuge ein opsi Paket
   Verläuft analog zu dem obigen Punkt 1 mit folgenden Unterschieden:
   Es werden zwei Setupprogramme für die Architekturen 32 und 64 Bit abgefragt und analysiert.
   Das Produkt bekommt ein zusätzliches Property: install_architecture mit den möglichen Werten:
   32bitonly, 64bitonly, both, systemspecific.

3. Analysiere einzelne Setup Datei
   erläuft analog zu dem obigen Punkt 1 nur das nach der Analyse des Setup-Programms
   abgebrochen wird.

4. Eine opsi Paketvorlage (Template) erzeugen
   Dieser Punkt fragt nicht nach einer Setup-Datei, sondern erstellt ein Template analog dem opsi-
   Produkt opsi-template nur das hier die Angaben aus der Produktkonfiguration bereits
   übernommen werden.

Opsi-setup-detector: Analyze single setup file and create an opsi package

The workflow is here described using Analyze single setup file and create an opsi package as an
example.
After you selected the task, you will get a file selection dialog. Select now the setup file that has to be analyzed. The analyze will start directly after the selection is done.

**Opsi-setup-detector: Analyze**

If the analyze found no result, you will get here a *Sorry unknown Installer*. If we have a successful analyze, you will see the result page.
Figure 175. opsi-setup-detector Result of the analysis

- **Detected Setup Type**: Type of detected Installer
- **MST allowed**: 
- **Link** with information about the installer
- **Setup file**: Path and name of the analyzed setup file
- **MST file**: For MSI installers or installers which contain MSI, an MST file can be specified here which will be integrated into the MSI call.
- **MsId**: For MSI installers or installers that contain MSI in the form of product code
- **Software version**: The version of the software to be installed if determinable.
- **Setup file size MB**: Size of the setup file in MB
- **Required space MB**: This value is an estimate of six times the size of the setup file and can be adjusted if necessary
- **InstallDir**: As far as detected the directory where the software will be installed.
- **Unattended installation command**: The determined command for a non-interactive installation.
- **Unattended deinstallation command**: The determined command for a non-interactive deinstallation.
- **Deinstallations program**: The determined deinstallations program

The values determined here can now be corrected or supplemented if necessary. The button Next Step leads to the first page of the product configuration. The metadata of the opsi product to be created is entered here.
The values determined here can be incorrect and are probably incomplete! After an initial installation, you should definitely check the values of **InstallDir**, **Check deinstallation program** and **software version** and adjust them in your script if necessary.

**Opsi-setup-detector: Product configuration 1**

![Opsi Setup Detector](image)

*Figure 176. opsi-setup-detector Product configuration 1*

- **opsi Product ID:** this is the name of the opsi package to be generated and is generated from the product name below, where spaces and other invalid characters are replaced by a -. The proposed opsi Product ID can of course be changed.

- **Product Name:** the name of the software to install. This may have to be corrected manually.

- **Product Version:** The version number determined from the name of the setup file must probably be corrected manually. It may only contain numbers and periods, since it's used for the versioning of the opsi package.

- **Description:** In this field the product name is given as default and should be completed with further hints, which are then set as product description of the opsi package.

- **License required:** If this checkbox is set, `$LicenseRequired$=true` will be set when patching the opsiscript.

**Opsi-setup-detector: Priority and dependencies**
For normal application software you don’t have to do anything here, due to the fact that the default settings fit. You can press the Next Step button.

Otherwise, here is an explanation of the settings that are possible:

**Priority**
- affects the installation order. **Recommended for application software: 0**
- Possible values are between 100 (at the very beginning) and -100 (at the very end). If product dependencies also exist, these will also additionally influence the installation sequence.

**Dependencies**
Here you can define the dependencies between products.

---

**Figure 177. opsi-setup-detector Product configuration 2**

**Figure 178. opsi-setup-detector Dependency Editor**
**Productid**

Productid (identifier) of the product to which a dependency exists.

**Dependency Mode**

You can either request the **Action** setup or (see below) the **Status** (installed).

**Action or Status**

For **Status**: Status that the product to which a dependency corresponds, should have (installed). If there is another status, the product is set to setup.

For **Action**: Action request, which should be set on the product, whereupon there is a dependency (setup).

**Dependency type**

Installation order. If the product for which there is a dependency must be installed before the installation of the current product, then this is before. If it must be installed after the current product, this is after. If the order doesn’t matter then nothing has to be entered here.

**Hint:**

Unfortunately there is currently no generic mechanism for uninstalling product dependencies. The Product Dependency mechanism is only reliable for action: setup and the (before- or after-) setup actions to be triggered and installed status, in the case of a required Action: uninstall this leads unfortunately to errors.

**Another hint:**

The actual installation order is determined by a combination of product dependencies and product prioritization. Details about this can be found in the opsi manual in the chapter *Manipulating the installation sequence by priorities and dependencies*.

**Opsi-setup-detector: Poperties**

Here, editable properties (product variables) can be defined for the product.
### Figure 179. opsi-setup-detector Property Editor

<table>
<thead>
<tr>
<th>Field / Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Name</td>
<td>Name of the product variable</td>
<td>This identifier is displayed in the product configuration in opsi-configed and can be read within the scripts with the function <code>GetProductProperty</code>.</td>
</tr>
<tr>
<td>Property Type</td>
<td>Variable type</td>
<td>Possible values: <code>Text</code> / <code>bool</code></td>
</tr>
<tr>
<td>Multivalue</td>
<td>Determines whether the product variable can take only exactly one or multiple values</td>
<td>Only available for type <code>Text</code></td>
</tr>
<tr>
<td>Editable</td>
<td>Determines whether the default values can be overwritten with new or additional values or not</td>
<td>Only available for type <code>Text</code></td>
</tr>
<tr>
<td>Description</td>
<td>Variable function description</td>
<td>Displayed as tooltip in opsi-configed</td>
</tr>
<tr>
<td>Possible values</td>
<td>Comma separated list of possible input values</td>
<td>If editable is set to “True”, the list can be added later within opsi-configed. Only available for type <code>Text</code>.</td>
</tr>
<tr>
<td>Default value</td>
<td>Default value</td>
<td>Selection list; Only available for type <code>text</code>: Free text field. Only available for type <code>Multivalue</code>: Multiple selection</td>
</tr>
</tbody>
</table>
Here you can select an icon to be displayed during the installation or you can accept the default icon (cogwheel) with Next step and switch to the next tab.

To select another icon, use the button Open icon directory to select the directory in which you expect to find icons. As a preselection you get a directory of open source icons: 128x128, supplied with the opsi-setup-detector. Select a subdirectory and the icons will be displayed. Now you can select an icon from the display.

After the product configuration is performed, the product can be created.

**Opsi-setup-detector: Create product**
Path to opsi-workbench is a drive letter or UNC path where the share `opsi_workbench` of your opsi-server is mounted.

For security reasons, an opsi package can only be rebuilt if it doesn’t exist already. If an existing package is to be overwritten, the directory must first be deleted from the opsi Workbench.

To the left of the button *Create opsi package* there are three possible options, which refer to the function of the button:

Create Mode' is a selection area where you can specify what happens when creating the package:

- **Create opsi product files** creates the directory tree for the new opsi package on the selected `opsi_workbench` if it does not already exist. The files required for the package will be created or copied.

- **Create opsi product files and build package** performs the operations described in the first point. Additionally, the opsi Package Builder is called to create the opsi package from the created directory tree. The exact processes are determined by the selection field *Tree mode*:
  - **Only build** starts the opsi Package Builder without interactive GUI, creates an opsi package from the directory tree via server command `opsi-makepackage` and terminates the opsi Package Builder after work is done.
  - **build and install** starts the opsi Package Builder without interactive GUI, creates from the directory tree via server command `opsi-makepackage` an opsi package installs the package via server command `opsi-package-manager` and finishes the opsi Package Builder after the work is done.

- **Create opsi product files and start interactive package builder** performs the operations listed in the first item. Additionally the opsi Package Builder is called interactively.

You have to quit it yourself to return to the opsi-setup-detector. For installation, configuration
and operation of the community project opsi Package Builder check

• Create opsi package is the button that initiates the package creation. For security reasons, an opsi package can only be rebuilt if it doesn’t exist already. If a package with this name already exists, you will be asked if the existing directory may be deleted.

Opsi-setup-detector: Analyze two setup files (32 and 64 bit) and create one opsi package

This item corresponds to the Analyze single setup file described above and create an opsi package with the following addition:
Two setup programs for the 32 and 64 bit architectures are queried and analyzed. The product gets an additional property: install_architecture with the possible values: 32bitonly, 64bitonly, both, systemspecific.

Opsi-setup-detector: Analyze single setup file

This item corresponds to the Analyze single setup file described above and create an opsi package with the following restriction:
After the analysis the process will be aborted.

Opsi-setup-detector: Create an opsi package template

This item corresponds to the Analyze single setup file described above and create an opsi package with the following restriction:
it does not ask for a setup file and therefore the analysis is not necessary. Rather a general script template is created.
## 9.21.7. Opsi-setup-detector: Configuration

![Picture of the configuration interface](image)

### Required configurations:

- **Name**
  (Used for entries in the changelog.txt)

- **Email**
  (Used for entries in the changelog.txt)

- **Path to the opsi-workbench**
  Path to the directory in which the opsi packages should be created. This is ideally the path to the place where the opsi_workbench of your opsi-server is mounted.

- **Path to the OpsiPackageBuilder**

Path to the OpsiPackageBuilder. OpsiPackageBuilder is used to build the opsi packages via ssh. see: [https://forum.opsi.org/viewtopic.php?f=22&t=7573](https://forum.opsi.org/viewtopic.php?f=22&t=7573)
Program behavior

• creatoradioindex
selects the Create mode Radiobutton

• buildadioindex
selects the Build mode Radiobutton

• registerinfilemanager
Should this program be registred to the FileManager (Explorer) context menu?

• workbench Mounted
Automatically detected. Is the opsi workbench reachable at workbench_Path.

Modification of the product:

• import libraries
List of opsi-script libraries that have to be imported.
One per line. May be empty. Example:

myinstallhelperlib.opsiscript

Configurations where you can enter opsi-script code for certain jobs:
opsi-script code lines which will be inserted.

List of opsi-script code lines that should be included.
One per line. May be empty. Example:

comment "Installation finished..."

• preinstall lines
opsi-script code lines that should be included before the installation starts.

• postinstall lines
Lines that should be included after the installation finished.

• preuninstall lines
opsi-script code lines that should be included before the uninstallation starts.

• postuninstall lines
opsi-script code lines that should be included after the uninstallation finished.

9.21.8. Opsi-setup-detector: Adaptation to your own requirements

The opsi-setup-detector is designed in such a way that it is as easy as possible to be continuously maintained and expanded. Certain customization options and help with problems are presented in the following. If you have ideas and/or wishes beyond that, please contact us at (info@uib.de) - we are looking forward to your suggestions.

Logging

The opsi-setup-detector creates log files under `c:\opsi.org\applog\opsisetupdetector.log`. Older log files are located in the same directory as `opsisetupdetector_0.log` to `opsisetupdetector_8.log`.

The log file becomes very large because it contains the complete data of the analysis of the setup file(s). For the analysis of the log file we recommend the `opsi-log-viewer` or another editor which can hide the unneeded log levels, because the text patterns found in the setup (on log level 8) in many cases are not so relevant.

Language support

When the program is started, the language of the Windows system is automatically determined. If a suitable language file is found for the language, it is used. If no supported language is found, English is used.

You can select the language via the menu item Languages or via the command line parameter `--lang=xx` where `xx` stands for the selected language.

Currently the opsi setup detector supports:

- German
- English
- Spanish

Additional languages can be easily added to be translated via a language file. Translations are done via the portal: https://www.transifex.com/opsi-org/

We look forward to your support

Unknown Installer

Surely you will come across the message Unknown Installer while working with the opsi-setup-detector. Then the opsi-setup-detector did not recognize the installer framework with which this installer was made.

Suggestions:

- Check if the file is really an installer. (Sounds stupid, but it happened to me already)
- Search the Internet for the keywords silent and product name
Once you have solved the problem, let us, (and the opsi community) know about experience.

The new (internal) design of the opsi-setup-detector makes it possible to easily add new installers.

What we need is:

- Example setup program
- Manufacturer or typical command line Switch for installation and uninstallation
- Informative links to the problem

**False detection**

Especially in the context of the increasing number of supported (i.e. detected) installers, false detection can occur.

This happens especially when an installer contains several software components which are packed with different installers.

Should you discover such a problem, please inform us and have the appropriate example and the correct solution ready.

**Program error**

If the opsi-setup-detector behaves strangely or does something unexpected, please backup the corresponding log file from `c:\opsi.org\appl\log` (see also: Section 9.21.9.1, “Logging”) and inform us about the problem at info@uib.de. Also have the sample file ready, as we may need it to reproduce the problem.

**Source Code and License**

The opsi-setup-detector is licensed under the AGPLv3.

The source code can be found under:

- https://gitlab.com/opsi-org
10. Troubleshooting

The following chapters describe ways to find and fix problems.

10.1. General troubleshooting

Usually we first see the symptoms of a problem and this is what makes us realise something went wrong. Our next step should be finding out where the problem lies. Once we know what our problem is we can attempt to fix it.

While checking where the problem comes from it is a good idea to revisit the relevant chapters and double check your configuration. Spending hours debugging a problem to then find out you made a typo can be frustrating.

When going through the following suggestions you should continuously try to reproduce the error to see if this made the problem disappear.

10.1.1. Install updates

Usually an easy first step is to make sure that you are using the latest versions when encountering problems. By switching to a newer version you increase the chance that an existing problem has been fixed in the meantime.

Whenever updates for opsi are released an information is posted on the forums. Along with the release information changelogs are posted that can help you identify what version of a component contains the fix relevant to your problem.

On a Debian-based system you can install updates for your OS with the following commands:

```
apt update
apt dist-upgrade
```

Afterwards you should update your opsi packages:

```
opsi-package-updater --v update
```

10.1.2. opsi configuration

You should make sure that your opsi server has the current configurations applied and that access rights are set as expected.

```
opsi-setup --set-rights
opsi-setup --init-current-config
```
If you are using the MySQL backend make sure that all migrations are applied:

```bash
opsi-setup --update-mysql
```

After this you should restart the opsi service so that they use the new configurations.

```bash
systemctl restart opsiconfd.service
systemctl restart opsipxeconfd.service
```

The configuration is applied now. You should check that the services are running:

```bash
systemctl status opsiconfd.service
systemctl status opsipxeconfd.service
```

In some cases the restart of a server may also work wonders.

### 10.1.3. Logging

To figure out what goes wrong a look into the logs is often the best route to follow. By default most opsi components log information that is useful during runtime but for finding out the cause of a problem you often have to increase the loglevel. After increasing the loglevel try to reproduce the previous behaviour and then look into the logs.

Once you finished solving your problem do not forget to reduce the loglevel back to defaults because high loglevel may slow down your servers or result in using up a lot of unwanted disk space.

### 10.2. Troubleshooting specific problems

The following chapters contain information about troubleshooting specific problems.

#### 10.2.1. MySQL server has gone away

This error is accompanied by an error message like to following:

```python
Execute error: (2006, 'MySQL server has gone away')
```

This error can have multiple causes which makes it important to look through the logs of your server.

**Packet size**

One possible cause of this error is that parts of queries exceed the allowed size. This is especially true if the error appears repeatedly over short time spans.
A solution to this can be to tune the `max_allowed_packet` setting of your MySQL server.

**Connection Pool: Connectin lifetime**

One possible cause of this can be that a connection of the connection pool of the MySQL backend is unable to reach the MySQL server after a long abstinence. In such a case it is possible that the server has closed the connection on its side.

A solution to this can be to limit the lifetime of connections by adjusting the value for `connectionPoolRecycling`. This should usually be lower than the timeout configured on the MySQL server (variable `wait_timeout`). With a default of 28800 seconds in MySQL 8.0 you could set the connection lifetime in opsi to 28500 seconds e.g.

### 10.2.2. Unable to access depot shares

Check if samba is running:

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

At least one `nmbd` and one `smbd` process should be running.

To restart Samba:

```bash
systemctl restart smbd.service
systemctl restart nmbd.service
```

It can also help to set the pcpatch-password. In a multi-depot environment we recommend to set this the same on all servers.

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

### 10.2.3. Accessing opsi webservice does not work

Check if the opsi service opsiconfd is running and responding.

Open the address `https://<server-address>:4447/info` with your browser. If you can’t connect check if the required services are running and check the corresponding logfiles.

If accessing the page works check the loads of the webservice. This makes use of `rrdtool` with Python bindinds. Install them if needed.

Make sure that enough space is available on the server.

If you can not connect, continue with next step. If you can connect: check the load of `opsi-webservice` and check the freespace on disk (scroll down in the info-page).
For the generation of load images you need *rrdtool* with Python-bindings. Please install them if needed.
11. opsi localization

11.1. Most opsi parts

The localization of the opsi software is done through Transifex: https://www.transifex.com/

Transifex allows for an easy translation of software projects.

You can find the opsi.org translation project at link: transifex.com/opsi-org/opsiorg/.

Everyone interested in translating opsi can participate!

With this we hope to get not only improved translations but also be able to integrate new languages that you are currently missing!

11.2. opsi configed

It is now possible to translate the opsi Configuration Editor - best know by its short name Configed - in Transifex. After logging in you will be able to access the resource configed.properties.

We think that translating Configed poses a bit more of a challenge. For most messages the context in which they appear is important and therefore simple word-by-word translation often does not result in a good translation.

This is why we want to handle things different in this case. In the past there were no reviews involved when we updated translations. But as Configed is such a vital and important part for many users that we will change our way from here on. To ensure the quality of translations for Configed that makes it of good and easy use for all opsi users we only want to publish translations that are reviewed.

To review a translation you need the role of a reviewer in Transifex. If you login as a user with this role you get a button in the translation interface that says review. Any reviewed string can not be further translated. If you are a partner, customer or someone very active inside the opsi community and you are interested in doing reviews for opsi please do not hesitate to contact us through mail or at the opsi forum.

11.3. Localization contact

Please send any questions, hints or localization files to: locales@uib.de
12. Glossary

**action request**
Next scheduled action for a client. Typical action requests are *setup*, *uninstall* and *update*.

**backend**
opsi supports different types of data storage, i.e. in a file or database. These are called backends.

**client Id**
Unique identifier for a client through a *fully qualified hostname* (FQDN).

**config server**
Server which is responsible for providing management and configuration data in an opsi environment.

**depot server**
Server which provides installation and process data through a Samba share for clients to install from.

**DHCP (Dynamic Host Configuration Protocol)**
A protocol allowing dynamic IP address assignment.

**FQDN (Fully Qualified Hostname)**
Complete name of a computer in a network. It consists of the IP name and domain, i.e. *fritzlaptop.uib.local*.

**FTP (File Transfer Protocol)**
A standard network protocol used for the transfer of files between a client and a server on a network

**GUI**
Graphical User Interface.

**host Id**
Unique ID of a host (includes servers and clients) by using the *fully qualified domain name* (FQDN).

**JSON**
JSON is short for JavaScript Object Notation, which is a compact data exchange format. The data is easy to read for people and for machines.

**Last seen**
Time stamp of the last time a client connected to the opsi service.

**LDAP (Lightweight Directory Access Protocol)**
Lightweight Directory Access Protocol is an protocol that allows programs to look up information
localboot product
An opsi product which is installed on a running operating system.

MAC address (Media Access Control address)
An unique identifier attached to the network adapter, which is transferred with every data package. With this address the computer (respectively its network card) can be identified worldwide and can be mapped to an IP address. The MAC address is composed from 6 hexadecimal numbers separated by colon.

netboot product
An opsi product which will be started when booting over the network. Usually used for installing operating systems on a computer.

opsi
open pc server integration

opsi-client-agent
Agent on a Windows client to install centrally managed software.

opsi-server

opsiconfd
Server-Dienst, welcher die opsi-Konfigurations-API als Webservice zur Verfügung stellt.

opsiconfd
Service on a server which provides the opsi configuration API as a webservice.

product properties
Additional parameters for an opsi product that can be set on a per-client basis and which are then evaluated during the installation.

product ID
Unique name of an opsi-product. Consists of a-z, numbers and hyphen. Spaces or special characters (except hyphens) are not allowed. Example for a product Id: acroread

product name
Full name of a product (allowing blanks). Example for a product name: Adobe Acrobat Reader.

PXE (Preboot eXecution Environment)
Standard for booting a software over the network. Commonly used with DHCP.
SAMBA
Free software to provide data shares or authentication services to Microsoft clients from an Linux server using the SMB protocol.

SMB (Server Message Block)
Protocol by Microsoft to provide network shares and authentication. A dialect of SMB is CIFS (Common Internet File System).

tftp (Trivial File Transfer Protocol)
A tftp daemon (tftpd) allows for transmitting files from and to other computers on a network without requiring a login but with restricted access. Accessing files is limited to those who are readable or writable by all users and located in a special directory.