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 Linux clients in opsi

This manual describes the operation of Linux clients in opsi.

It's assumed that the installation and startup of an opsi-server has already been performed.

Essential topics of this manual:

1. Automatic Linux OS installation
2. Enrollment and integration of Linux computers in opsi (Installation of the opsi-client-agent)
3. Provision of the standard opsi software for Linux on the opsi server
4. Installation of standard software on the Linux clients
5. Standard opsi software for Linux under opsi
6. Packaging your own software
7. Creation of opsi packages
8. Notes on Linux clients
   a. Special commands for Linux

2.1. Conventions of this document

Commands are highlighted separately:

```
this is a command
```

As part of the installation and configuration process, you can copy and execute the commands from these fields in order by copy & paste from this document.

Commands or file names will be highlighted as: `opsi-set-rights` oder `/Applications/opsi-script`.

This is an opsi-script code:

```
Message "Installing "+$ProductId$+" ...
```

Chapters containing the name of a particular platform are specific to that platform. The supported platforms are:

- Windows
- Linux
- macOS
3. Requirements for Linux clients

In the following the requirements for the management of macOS clients under opsi are described.

The opsi-mac-client-agent is a opsi extension. This means that you need an unlock file to use it. You get this unlock file when you buy the extension. For evaluation purposes we also provide a time limited unlock file for free (→ mail to info@uib.de).


Technical requirements is an opsi-server with opsi 4.1. or higher.

### 3.1. Linux versions supported as opsi client

<table>
<thead>
<tr>
<th>Distribution</th>
<th>OS-Installation</th>
<th>netboot products</th>
<th>client-agent</th>
<th>opsiclientd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian 11 Bullseye</td>
<td>✔</td>
<td>debian, debian11</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Debian 10 Buster</td>
<td>✔</td>
<td>debian, debian10</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Debian 9 Stretch</td>
<td>✔</td>
<td>debian, debian9</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Debian 8 Jessie</td>
<td>▼</td>
<td>debian, debian8</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Ubuntu Bionic 20.04 LTS</td>
<td>✔</td>
<td>ubuntu, ubuntu20-04</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>Linux Mint 20-3</td>
<td>✔</td>
<td>mint20-3</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Linux Mint 20-2</td>
<td>✔</td>
<td>mint20-2</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Linux mint 20-1</td>
<td>✔</td>
<td>mint20-1</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>RHEL 8</td>
<td>✔</td>
<td>rhel8</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>Alma Linux 8</td>
<td>✔</td>
<td>alma8</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Rocky Linux 8</td>
<td>✔</td>
<td>rocky8</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CentOS 8</td>
<td>▼</td>
<td>centos8</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 SP3</td>
<td>✔</td>
<td>sles15sp3</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SLES 15 SP2</td>
<td>✔</td>
<td>sles15sp2</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SLES 15 SP1</td>
<td>✔</td>
<td>sles15sp1</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>SLES 12 SP5</td>
<td>✔</td>
<td>sles12sp5</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>
</tbody>
</table>
### 3. Requirements for Linux clients

<table>
<thead>
<tr>
<th>Netbootproduct</th>
<th>Installer</th>
<th>State</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>debian</td>
<td>opsi</td>
<td>✔</td>
<td>stretch - bullseye</td>
</tr>
<tr>
<td>debian11</td>
<td>distribution</td>
<td>✔</td>
<td></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>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 - jammy</td>
</tr>
<tr>
<td>ubuntu22-04</td>
<td>distribution</td>
<td>✔</td>
<td></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>ubuntu14-04</td>
<td>distribution</td>
<td>▲</td>
<td></td>
</tr>
<tr>
<td>mint20-3</td>
<td>distribution</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>mint20-2</td>
<td>distribution</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>mint20-1</td>
<td>distribution</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>centos8</td>
<td>distribution</td>
<td>▲</td>
<td></td>
</tr>
<tr>
<td>centos70</td>
<td>distribution</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>redhat8</td>
<td>distribution</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>redhat70</td>
<td>distribution</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>sles15sp3</td>
<td>distribution</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>sles15sp2</td>
<td>distribution</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>sles15sp1</td>
<td>distribution</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>
## 3. Requirements for Linux clients

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>sles12sp4</td>
<td>✔️</td>
</tr>
<tr>
<td>sles12sp3</td>
<td>✔️</td>
</tr>
<tr>
<td>sles12sp2</td>
<td>✅</td>
</tr>
<tr>
<td>sles12sp1</td>
<td>✅</td>
</tr>
<tr>
<td>sles12</td>
<td>✅</td>
</tr>
<tr>
<td>opensuse15-3</td>
<td>✔️</td>
</tr>
<tr>
<td>opensuse15-2</td>
<td>✅</td>
</tr>
<tr>
<td>opensuse15-1</td>
<td>✅</td>
</tr>
<tr>
<td>opensuse15</td>
<td>✅</td>
</tr>
<tr>
<td>opensuse14-3</td>
<td>✅</td>
</tr>
<tr>
<td>opensuse14-2</td>
<td>✅</td>
</tr>
<tr>
<td>opensuse14-1</td>
<td>✅</td>
</tr>
<tr>
<td>ucs50</td>
<td>✔️</td>
</tr>
<tr>
<td>ucs44</td>
<td>✔️</td>
</tr>
<tr>
<td>ucs43</td>
<td>🔴</td>
</tr>
</tbody>
</table>
4. Installing the minimal Linux opsi products

To distribute software with opsi, ready-made products are available for installation. These include the agent ('opsi-client-agent'), which must be installed on the clients for management.

There is an automated and a manual way to do this. The automated option is recommended.

4.1. Automatic installation of the minimal Linux opsi products

For the automatic installation of the opsi products there is the tool `opsi-package-updater`, which configured as in '/etc/opsi/opsi-package-updater.conf' or `/etc/opsi/package-updater.repos.d/`, automatically fetches the current packages from the opsi repository and installs them on the server.

The configuration of the opsi repository for Windows clients can be found in the directory `/etc/opsi/package-updater.repos.d/` in the file `uib-linux.repo`.

Activate the desired repository by setting the entry `active = true` in the desired *.repo file.

### Listing 1. /etc/opsi/package-updater.repos.d/uib-linux.repo

```ini
[repository_uib_linux]
description = opsi Linux Support
active = true
baseUrl = http://download.uib.de
autoInstall = false
autoUpdate = true
autoSetup = false
proxy =

; Set Proxy handler like: http://10.10.10.1:8080
```

Install the packages on the server by running this command as root:

```
opsi-package-updater -v --repo uib_linux install
```

After a successful installation you have to reload all data in `opsi-configed` so that the new products are visible there.

If the connection to the Internet has to be routed via a proxy, this must be entered in the `.repo` configuration files in `/etc/opsi/package-updater.repos.d/` as the value for `proxy`. From version 4.1.1.33 of opsi-utils a global proxy can be configured in `/etc/opsi/opsi-package-updater.conf`.

```ini
[repository_uib_linux]
...
proxy =
```

If the installed packages are to be updated later, this can be linuxed with the following command:
Further information on `opsi-package-updater` can be found in the manual.

4.2. Manual installation of the Linux opsi products

There is also the option of downloading and installing the packages manually.

Get the current opsi packages in the `.opsi`-package format. You can find the packages at https://download.uib.de/opsi4.2/stable/packages/linux/localboot

We recommend saving the `.opsi` files under `/var/lib/opsi/repository`. To ensure that the process `opsiconfd` can access the files, you should run `opsi-set-rights /var/lib/opsi/repository`.

After downloading you have to install the packages on the server with the command `opsi-package-manager -i <packagename>.opsi`. 
5. Adding clients to opsi

To be able to manage computers with opsi, they must be known to the opsi system. In addition, an agent must be running on these computers so that communication between the server and client is possible. No management is possible without this client agent.

Depending on the environment in which opsi is to be used, there are different procedures. If there are already clients in the environment with an installed operating system that are to be managed with opsi, they can be integrated in different ways.

The alternative to this is that the computers to be managed by opsi are equipped with a new operating system. As part of the installation of the operating system, the required agent is also installed by opsi. However, any previously installed software (including the operating system) will be removed. To use this procedure you first add a client to opsi and then perform an OS installation.

5.1. Creation of a new opsi client

To manage computers, they must be known to the opsi-server. This chapter describes different ways to create a client in opsi for later management. This is particularly helpful if you want to install an operating system on your computer using opsi.

For the integration of clients with an already installed operating system, please read the chapter integration of existing Clients.

5.1.1. Creating a new opsi client via the graphical management interface

A client can be added to the opsi-server through the opsi-configed graphical user interface.

From the menu, choose OpsiClient / Create new opsi client and enter:

- Client name
- DNS domain (if different from the default)
- Client description
- IP address (required if DNS can not be used resolve the address of the client)
- MAC address (required if the opsi-server is the DHCP server or if you want to use PXE boot with this client)

After completing the input, the client will be created on the opsi-server, and if the opsi-server is also the DHCP server, the client will also be created in the DHCP configuration, as a PXE client.

The list of configured opsi clients can be viewed at any time in the opsi-configed mode Client configuration under the clients tab.

5.1.2. Creating a new opsi client via the command line

A client can added through the command line using the tool opsi-admin.

The syntax is the following:

```
opsi-admin -d method host_createOpsiClient <client-id> [opsiHostKey] [description] [notes] [hardwareAddress] [ipAddress] [inventoryNumber] [oneTimePassword] [created] [lastSeen]
```

Missing values usually use a default value - most fields are then empty.

The following command will create the client testclient.domain.local with a random host key, the description Testclient, no notes, the MAC address of 00:0c:29:12:34:56 and the IP address 192.0.2.1:
5.1.3. Creating a new opsi client using the opsi-client-bootcd

On the download page of uib you will find various ISO images of the 'opsi-client-boot-cd' at https://download.uib.de/opsi4.2/boot-cd/. Download the latest and burn it to a CD.

Start the computer from the CD. You then should see the following screen:

![Start image opsi-client-boot-cd](image)

**Figure 1. Start image opsi-client-boot-cd**

Choose *Start opsi (English)*. After a while, the following screen will appear. If your DHCP server assigns IP addresses to unknown DHCP clients, then most fields will already have valid values. Otherwise you have to complete the missing data by hand. You must at least give the hostname.
Then choose OK.

Then choose Admin account. This tells the client to register itself at the opsi-server using provided credentials.

Now you will get a login window, where you must authenticate yourself as a member of the opsiadmin group. If this was successful, then the client sends its data to the server, at which point the client will be created at the server. In the next step, the client asks the server for the list of available netboot products, and makes them available for you to choose from.
Now you may choose the operating system that you would like to install (or e.g. hwinvent).

### 5.2. Integration of existing Linux clients into opsi.

To include existing Linux clients in opsi, the opsi-client-agent must be installed on them. This can be performed in several ways. After you have installed the opsi-client-agent, as described below, the client will also appear in the client list of the opsi-configed, in the case you had not already added it there previously.

Basically, is possible to **run on the client** or **from the server** to trigger the installation of the agent.

Running the installation directly on the client is appropriate for individual machines. For a mass deployment of the agent, the `opsi-deploy-client-agent` is generally more suitable. If the necessary unlocks are available on the Linux clients.

If there is already another way to deploy software, it's also possible to deploy the opsi-client-agent and run the `silent_setup.sh` script included in the package.

Once the agent is installed, existing opsi products can be **installed** on these clients.

#### 5.2.1. Using opsi-client-agent-installer on Linux

1. Logon to the client.
2. Download the installer from your configserver. It is located at `https://<fqdn_or_ip_of_the_configserver>:4447/public/opsi-client-agent/` and has the file name:
   - Windows: opsi-client-agent-installer.exe
   - Linux: opsi-linux-client-agent-installer.sh
   - macOS: opsi-mac-client-agent-installer.sh
1. Execute the installer (for Linux and MacOS this must be done with root-rights, on Windows a UAC-Request may be displayed).

2. The installer will extract itself into a temporary local directory and start the oca-installation-helper.

This shows a user interface with input fields for Client-ID, Opsi Service URL, Username and Password. The fields are pre-filled (if possible e.g. if an old opsicliend.conf is found), but you may need to add or change some of the data.

- **Client-ID** should be the fqdn of the Client.
- **Opsi Service url** should have the format `https://<fqdn_or_ip_of_the_configserver>:4447`.
- **Username and Password** should correspond to a user of the group `opsiadmin` in case of a first installation. For reinstallation it is also possible to use Client-ID and pckey for authentication.

After starting the Installer by clicking the button **Install** the installer connects to the server to register the client at the server. Afterwards the installer calls the included opsi-script to execute the setup.opsiscript of the opsi-[linux-|mac-]client-agent.

If the installation is finished the installer terminates.

Further information around the opsi-client-agent Installer and the command line parameters and other possibilities to install the opsi-client-agent you will find at the opsi-manual in the chapter **Subsequent installation of the opsi-client-agents**

5.2.2. Using service_setup.sh on Linux (outdated)

The method described over here is only for backward compatibility to opsi 4.1 and the corresponding opsi-client-agent versions 4.1. Please use as possible the opsi-client-agent Installer.

- Log in to the client.
- Start the terminal program
- For the following commands you need to replace the following placeholders:
  - `<username>` with your login name.
  - `<mnt>` with a directory name that does not exist yet e.g. 'mnt'.
  - `<serviceuser>` with a username that is known on the opsi-server.
  - `<servicepass>` with the password of the `<serviceuser>`. You can also omit `pass=<servicepass>`, then you will be prompt to input the password
  - `<opsi-server>` the name or IP number of the opsi-server.

```bash
sudo su
cd /mnt
mount.cifs //<opsi-server>/opsi_depot /mnt -o user=<serviceuser>
cd /mnt/opsi-linux-client-agent
./service_setup.sh
cd
umount /mnt
```

without password query

```bash
sudo su
cd /mnt
mount.cifs //<opsi-server>/opsi_depot /mnt -o user=<serviceuser>, pass=<servicepass>
cd /mnt/opsi-linux-client-agent
./service_setup.sh
cd
umount /mnt
```

Example:

```bash
sudo su
cd /mnt
mount.cifs //sepia/opsi_depot /mnt -o user=adminuser
cd /mnt/opsi-linux-client-agent
./service_setup.sh
cd
umount /mnt
```

1. Start from the mounted share the script `opsi-linux-client-agent\service_setup.sh`
   Confirm with 2 x Enter
2. The script copies the necessary files into a temporary local directory and then starts opsi-script for the actual installation.
3. The script contacts the server via opsi webservice to create the client on the server side and to find out the pckey. This is done first with the user/password combination entered in `config.ini`. If this fails, a login window appears with service URL (opsi-configserver), user name and password. Here a user is needed which is a member of the group 'opsiadmin'. It is possible to also operate with a user which is only allowed to execute the method `host_createOpsiClient`. 
The client needs a reboot after the installation.

## 5.2.3. Using opsi-deploy-client-agent for Linux.

The `opsi-deploy-client-agent` script deploys the opsi-client-agent directly from the opsi-server to the clients. It's easy to integrate a large number of clients from the server into an opsi environment. As a prerequisite for the clients is needed:

- an activated ssh access
- the root account with password

The `opsi-deploy-client-agent` script can be found at `/var/lib/opsi/depot/opsi-client-agent`.

Run the script with 'root' privileges. If the script is not executable, you can fix this problem with the following command:

```bash
```

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

The installation runs in the background without any interaction from user and transparently.

The command `opsi-deploy-client-agent` has several call parameters.

All following examples assume that you have switched to the root directory of the opsi-client-agent product:

```bash
cd /var/lib/opsi/depot/opsi-linux-client-agent
```

Typical calls are:

- For a single client:

  ```bash
  ./opsi-deploy-client-agent -v --user=root mylinuxclient.local
  ```

  Results in the following output:

  ```
  Password is required for deployment.
  Password:
  [5] [2021-02-04 16:43:43.121] [ ] Starting deployment to host mylinuxclient.locall (posix.py:84)
  [5] [2021-02-04 16:43:43.121] [ ] Querying for ip address of host mylinuxclient.locall (common.py:158)
  [5] [2021-02-04 16:43:43.122] [ ] Got ip address 192.168.10.70 from syscall (common.py:167)
  [5] [2021-02-04 16:43:43.123] [ ] Pinging host 192.168.10.70 ... (common.py:183)
  [5] [2021-02-04 16:43:44.146] [ ] Host 192.168.10.70 is up (common.py:194)
  [5] [2021-02-04 16:43:44.153] [ ] Patching config.ini (posix.py:91)
  [5] [2021-02-04 16:43:44.157] [ ] Copying installation scripts... (posix.py:107)
  [5] [2021-02-04 16:43:48.316] [ ] Running installation script... (posix.py:147)
  [5] [2021-02-04 16:43:53.382] [ ] opsi-client-agent successfully installed on mylinuxclient.locall (posix.py:176)
  [5] [2021-02-04 16:43:53.395] [ ] Restarting opsiclientd service on computer: uib-mmini1 (posix.py:331)
  [5] [2021-02-04 16:43:55.620] [ ] 1/1 deployments successfully (__init__.py:210)
  ```

- For a list of clients:

  ```bash
  ./opsi-deploy-client-agent -v --user=root --hosts-from-file HOSTFILE.TXT --skip-existing-clients
  ```

Here HOSTFILE.TXT is a file with one client name (FQDN) per line. As long as the clients are not known to the opsi-server, it tries to install the `opsi-mac-client-agent` on this machine.
5. Adding clients to opsi

- Display all command line parameters:

  ./opsi-deploy-client-agent --help
6. Rollout of existing products to Linux.

For the deployment of software to clients, the *opsi-client-agent* must be installed on them. This can be rolled out on existing machines.

Subsequently, the *opsi-configed* management interface is used to deploy software to clients.

The following products are provided by opsi for Linux as standard:

- `opsi-linux-client-agent`
- `swaudit`
- `hwaudit`
- `l-system-update`
- `opsi-configed`
- `opsi-logviewer`
- `opsi-auto-update`
- `opsi-linux-client-kiosk`
- `opsi-setup-detector`

6.1. Deploying opsi standard products: *opsi-configed*

One of the opsi standard products is the product *opsi-configed*, which installs the opsi Management Interface. This Application is a Java application, therefore a Java Runtime Engine is bundled with the product.

Using *opsi-configed*, in the mode *Configuration of clients*, choose the appropriate client in the tab *Clients*.

If you have not already done so, update the data of *opsi-configed* by using *File / Reload all data* or click the reload icon.

Switch to the tab *Product configuration*, look for the line with the product *opsi-configed*. Click in the column *Requested Action*, and select the action *setup*.

The check mark in the icon menu bar should change its color to red. If you click on it, the new settings will be transmitted to the opsi-server, afterwards its color will be green again.

Reboot the client. The opsi-client-agent should start and install the product *opsi-configed*. After the installation you can find *opsi-configed* in the start menu.

6.2. Inventory with the localboot products *hwaudit* and *swaudit*.

In *opsi-configed*, *Client configuration* mode, under the *Clients* tab, select the client under consideration.

If not already performed, update the opsi-configed’s dataset using *Reload File/Data* or clicking the corresponding icon.

Go to the *Product configuration* tab, click in the *Requested* column for the *hwaudit* product, this will open a list/dropdown menu and there select the *setup* action. Repeat this for the *swaudit* product.

The check mark in the icon menu bar should change its color to red. If you click it, the new settings will be transmitted to the opsi-server, afterwards its color will be green again.
Then restart the client. It should now start the `opsi-client-agent` and install the `hwaudit` and `swaudit` products. With `hwaudit` and `swaudit`, hardware and software information, respectively, is collected and transmitted to the opsi-server. The collected information is displayed under the `Hardware Information` and `Software Inventory` tabs, respectively.
7. Management interface opsi-configed

Opsi offers with the opsi-configed a comfortable management interface. It communicates via HTTPS with the opsi server and can therefore be used on any computer that can establish a corresponding connection.

When using a virtual machine, make sure that the virtual screen has a large enough resolution. For opsi-configed a minimum resolution of 1024x768 pixels is required. To improve the graphics and mouse driver integration at a higher resolution, it is helpful to install the ‘VMware Tools’ on a VMware machine or the virtual guest additions on a VirtualBox machine.

7.1. Installation of the management interface opsi-configed

The management interface is installed as a local application on the administration PCs. In your web browser, go to the address https://<opsidepotserver>:4447/. There you will find links to installers for different operating systems.

Alternatively, you can find corresponding installers under https://download.uib.de/opsi4.2/misc/helper/.

The Windows installer must be executed with administrative rights. To do this, right click to open the context menu of the installer and then select ‘Run as administrator’.

Once one PC is equipped with the management interface, further PCs can have easily have the interface Section 6.1, “Deploying opsi standard products: opsi-configed” installed with the localboot product opsi-configed, as long as the opsi agent is already installed on the PC.

7.2. Start of the management interface opsi-configed

Start opsi-configed via the shortcut in your Start menu.

Log in as a user who is a member of the group opsiadmin.

The operation of the management interface is pretty much self explanatory. You will find detailed instructions in the opsi manual.

Changes in the opsi management interface must be saved before they take effect and changes in the data must be retrieved from the server via the ‘Reload data’ button.
8. Hardware Inventory with the netboot product hwinvent

If the product 'hwinvent' is installed on your opsi server and you have added a client Section 5.1, “Creation of a new opsi client” which is configured to boot over the network, you can do something else useful: Hardware inventory when there is no operating system installed.

Using 'opsi-configed', in the mode 'Configuration of clients', choose the appropriate client in the tab 'Clients'. If you have not already done so, update the data of opsi-configed by using 'File / Reload all data' or click the reload icon. Switch to the tab 'Netboot products', look for the line with the product hwinvent. Click in the column 'Requested Action', and select the action 'setup'. The check mark in the icon menu bar should change its color to red. If you click on it, the new settings will be transmitted to the opsi-server, afterwards its color will be green again.

Then reboot the client. It should now pull a Linux image over the network (via PXE), to scan the hardware of the PC and then reboot it. If the computer was not otherwise already set up, afterwards the message appears that no operating system is installed on the disk.

The results of the hardware scan have been transmitted to the opsi-server. The results can be viewed under the 'Hardware information' tab.

In case the screen remains black after booting the bootimage or if the network card does not work (correctly), the start parameters of the bootimage may have to be adjusted for this specific hardware. This can be achieved using 'opsi-configed' in the 'Host parameters' tab by editing the entry 'opsi-linux-bootimage.append'.

More information can be found in the opsi manual, in the chapter 'netboot products'.

In case the screen remains black after booting the bootimage or if the network card does not work (correctly), the start parameters of the bootimage may have to be adjusted for this specific hardware.

This can be achieved using 'opsi-configed' in the 'Host parameters' tab by editing the entry 'opsi-linux-bootimage.append'.

More information can be found in the opsi manual, in the chapter 'netboot products'.

In case the screen remains black after booting the bootimage or if the network card does not work (correctly), the start parameters of the bootimage may have to be adjusted for this specific hardware.

This can be achieved using 'opsi-configed' in the 'Host parameters' tab by editing the entry 'opsi-linux-bootimage.append'.

More information can be found in the opsi manual, in the chapter 'netboot products'.
9. Installation of a new Linux PC via opsi (OS-Installation)

9.1. opsi Linux netboot products based on the distribution installer

- 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.
- The installation of Linux Mint uses the content of the official installation media provided via an NFS share.
- 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, RedHat and Linux Mint 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.

9.1.1. Providing the installation media on the server

For Suse, RedHat and Linux Mint 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/opensuse15-2-64/openSUSE-15.2-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.
  However the Linux Mint installation requires the installation media content, including hidden directories, to be copied into the specific `opsi_nfs_share` subdirectory.
- 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 product `ubuntu22-04` a `.iso` file has to be placed in the `iso` directory within the package directory in the depot.
**Start order of involved services for SLES 12**

It may occur that the `showmount` command exits with an error message like the following:

```bash
# 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:

```bash
# systemctl restart rpcbind.service
# systemctl restart nfs-server.service
```

Then showmount works as expected:

```bash
# showmount -e localhost
Export list for localhost:
/var/lib/opsi/depot/opsi_nfs_share *
```

---

### 9.1.2. Common properties of the Linux netboot products based on the distribution installer

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:** (not present in ubuntu22-04!)
  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='')
9.1.3. The products debian9, debian10, debian11 and ubuntu18-04, ubuntu20-04, ubuntu22-04, mint20-1, mint20-2, mint20-3

The basic OS installation files are taken from the distribution repositories.

This product has the productive state.

This product has the following additional properties, some are not present in the netboot package ubuntu22-04:

- **online_repository:**
  distribution repository for the installation. (only for the Debian/Ubuntu family) (Default=distribution dependent)

- **encrypt_password:** (not present in ubuntu22-04!)
  Password for disk encryption (only if encrypt_logical_volumes=true)
  Example: `linux123` Default: `linux123`

- **partition_disk:** (not present in ubuntu22-04!)
  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 (direct in ubuntu22-04) / lvm: use LVM to partition the disk / crypto: use LVM within an encrypted partition (not present in ubuntu22-04!)
  Possible: “regular” or “direct”, “lvm”, “crypto”
  Default: `lvm`

- **partition_recipe:** (not present in ubuntu22-04!)
  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`

- **installation_method:**
  Offers a possibility to bypass the kexec in case it fails. The alternate method installs a minimal installation on the local disk and reboot from it. This property currently works with `debian10`, `ubuntu18-04` and `ubuntu20-04` in UEFI. Possible: “kexec”, “reboot”
  Default: `kexec`

**Videos (time lapse)**


**The product ucs44**

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:
9. Installation of a new Linux PC via opsi (OS-Installation)

- **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`

### 9.1.4 The products sles12, sles12sp1, sles12sp2, sles12sp3, sles12sp4, sles15, sles15-1, sles15-2

This product has the following additional properties:
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:xxxxxxxxxxxxxx"]
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/sles12sp2/64/ 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.


9.1.5. The products redhat70, redhat8 and centos70, centos8, alma8, rocky8
This product has the following additional properties:
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:xxxxxxxxxxxxxx"]
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/centos8/64/ 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/redhat8/64/ 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.


9.2. Linux netboot products with generic installer

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 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.

### 9.2.1. Common properties of Linux netboot products (generic installer)

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')

- **language:**
  language / locale to be installed (default='de')

- **timezone:**
  time zone to be configured (default='Europe/Berlin')

- **root_password:**
  root password (default='linux123')

- **user_password:**
  user password (default='linux123')

- **online_repository:**
  repository to use for installation - repository of the Linux distribution to be used for installation (not for SLES) (default = distribution dependent)

- **proxy:**
  proxystring (if required) as: 'http://<ip>:<port>' (default='')

- **additional_packages:**
  additional packages to install. Packages names separated by blanks. (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)
9.2.2. Ubuntu

The basic installation is performed per debootstrap directly from the network.

This product has the status 'productive'.

It is UEFI/GPT compatible.

9.2.3. Debian

The basic installation is performed per debootstrap directly from the network.

This product has the status 'productive'.

It is UEFI/GPT compatible.

For this product applicable opsi-server packets are available, that can be installed by setting 'install_opsi_server=true'.
10. Integration of own software into the software distribution of opsi.

The installation of software in opsi is performed by the opsi-client-agent and specially by the script controlled setup program opsi-script. Therefore an opsi-script script must be created for each opsi product. Afterwards this script, as well as the installation files and metadata are packed into an opsi product, which can finally be installed on the opsi-server.

10.1. A small tutorial on how to create an opsi-script script.

10.1.1. Introduction

This tutorial merely helps you getting started with opsi. It can’t replace professional training (which you may order through uib), or thoroughly studying the complete opsi manuals (which might be time consuming and partially error prone if you lack background knowledge). uib now offers training in English, too.

Training and Support:
Get Training by uib gmbh in Europe or possibly Northern America:
https://uib.de/en/support-training/support/

Manuals:
The opsi Manuals can be found at: https://uib.de/en/opsi-documentation/documentation/ important for scripting:
opsi-script reference card and opsi-script manual

Wiki (Scripts, Tips, Links):
https://forum.opsi.org/wiki

Support Forum (fast and free vendor support):
https://forum.opsi.org

10.1.2. Methods of non-interactive software installation on Linux

Linux (unlike Microsoft) standardized its software installation methods very early on. The problem over here is that these standards differ between the distributions. Here are the core variants to install software on Linux:

- Install a package from a repository
- Install a package from a file (*.rpm, *.deb)
- Install a package with a third party installer
- Install a package from the source code (make install)

In the first two cases a unattended installation is no problem.

Sometime you will get linux software in packed formats like *.zip, or *.tgz.

All variants mentioned so far can be installed directly by opsi-script, except *.tgz which must be unpacked before.
10.1.3. Structure of an opsi-script script

In the subsequent chapters the basic elements of an opsi-script will be described with examples for Windows.

First an example for a simple opsi-script script:

```plaintext
[Actions]
WinBatch_tightvnc_silent_install

[WinBatch_tightvnc_silent_install]
"%ScriptPath%\tightvnc-1.3.9-setup.exe" /silent
```

An opsi-script script consists of **primary** and **secondary** sections. Sections are introduced with a section name in square brackets, as known from the ini files.

The true software installation work takes place in the secondary sections, which are called by the primary sections.

The secondary sections are “theme specific” and each has a special syntax.

The section name of a secondary section starts with its type, followed by a freely determinable name.

In the shown example, the primary section [Actions] calls a secondary section [WinBatch_tightvnc_silent_install].

The secondary section is of the type WinBatch. The content of a WinBatch section is executed via the Windows API.

So in this case the setup program `tightvnc-1.3.9-setup.exe` is started with the parameter `/silent`.

10.1.4. Primary Sections

**Actions**

The [Actions] section is the actual main program. This is where the script processing begins.

**Sub-sections**

Program sections that are required constantly can be redistributed to sub sections (subroutines). It’s possible to source sub sections to external files.

The primary sections are the main program in which the script flow is controlled. For this there are:

- Variables: Strings and string lists
- if elseif else endif statements
- for loops over string lists
- Functions
10. Integration of own software into the software distribution of opsi.

10.1.5. Important secondary sections

Files

File operations, such as:

- copy (with version control, recursive ...).
- delete
- create directories
- ...

WinBatch

Used to call programs via the Windows API. For example, invokes to setup programs in silent mode are made in these sections.

ShellInAnIcon

The content of this section is passed to the operating system on the typical shell for execution. This shell is the cmd.exe for Windows, for Linux and for macOS the bash. As such, normal batch scripts can be stored here.

Name variants of ShellInAnIcon with identical behavior are Shellbatch, DOSBatch and DOSInAnIcon.

ExecWith

The contents of these sections are passed to an external program such as an (interpreter) for execution. For example, ExecWith can be used to integrate AutoIt scripts http://www.autoitscript.com directly into the opsi-script script.

Registry

The Registry sections are used to edit the registry.

LinkFolder

LinkFolder sections are used to create and remove shortcuts. For example, such shortcuts can be created on the desktop or in the start menu.

10.1.6. Global constants

Global constants are text placeholders that can be used in primary and secondary sections and are textually replaced by their values at runtime.

The use of placeholders can be utilized to ensure that paths are set correctly in different environments (in the case of systems with different languages or operating system versions for example).
10. Integration of own software into the software distribution of opsi.

Examples:

- `%ProgramFiles32Dir%`  
  `c:\Program Files (x86)`

- `%Systemroot%`  
  `c:\windows`

- `%System%`  
  `c:\windows\system32`

- `%opsiTmpDir%`  
  `c:\`

- `%Scriptpath%`  
  `<path to running script>`

### 10.1.7. Second example: tightvnc

For clarification purposes, now a simple script for the installation of `tightvnc`. As a matter of fact this script would get on with the call of the silent installation in the *Winbatch section*. However, during a repeated installation an interactive dialog appears here (because of the restart of the running service). This dialog window is closed (if it appears) with the help of 'AutoIt'.

```
[Actions]
Message "Installing tightvnc 1.3.9 ...
ExecWith_autoit_confirm "%ScriptPath%\autoit3.exe" WINST /letThemGo
WinBatch_tightvnc_silent_install
KillTask "autoit3.exe"

[WinBatch_tightvnc_silent_install]
"%ScriptPath%\tightvnc-1.3.9-setup.exe" /silent

[ExecWith_autoit_confirm]
; Wait for the confirm dialog which only appears if tightvnc was installed before as service
; Waiting for the window to appear
WinWait("Confirm")
; Activate (move focus to) window
WinActivate("Confirm")
; Choose answer no
Send("N")
```

### 10.1.8. Elementary commands for primary sections

**String-Variable**

**Variable declaration**

`DefVar <variable name> [= <initial value>]`

**Variable assignment**

`Set <variable name> = <value>`

**Example:**
String variables are handled differently in primary and secondary sections. In primary sections, string variables are independent objects. Only here they can be declared and assigned values. Accordingly, the connection of variables and strings to a string expression is to be performed with a "+" operator.

Example: "Installing "$ProductId$" ...

In secondary sections, string variables are replaced with the contents of the variable before the section is executed.

For example: "Installing $ProductId$ ..."

This should be taken into consideration when the corresponding string expressions are cut and pasted in the script.

The advantage of this construction is that in sections that are executed outside the 'opsi-script' (DosBatch / Execwith) opsi-script variables can be manipulated at ease.

Message / ShowBitmap

Text output during installation:

Message <string>

Example:

Message "Installing "$ProductId$" ..."

To output a graphic during the installation:

ShowBitmap <filename> <subtitle>

Example:

ShowBitmap "%ScriptPath%\python.png" "Python"

if [elseif] [else] endif

Syntax:

if <condition>
  ;statement(s)
elseif <condition>
  ;statement(s)
else
  ;statement(s)
endif
Functions

HasMinimumSpace
Checks for free space on the hard disk.

FileExists
Checks for the existence of a file or directory.

Errors, logging and comments

Comment characters ‘;’
Lines that start with a semicolon (‘;’) are not interpreted.

Comment
Writes a comment message to the log file.

LogError
Writes an error message to the log file.

IsFatalError
 Cancels the execution of the running script and reports the installation as failed.

Condition for execution

requiredWinstVersion
specifies the (minimum) required opsi-script version.

Other important opsi-script functions

An overview of the opsi-script functions is given by the reference card

A detailed documentation can be found in the opsi-script manual:

Here are a few more notes on particularly important elements:

Stringlists:
String lists are very powerful, especially for evaluating output from external programs. Read the opsi-script docs for more information.

ExitWindows:
Reboot/Shutdown the system and finish the opsi-script.

- ExitWindows /Reboot
  Computer restart after completion of the running script.
- ExitWindows /ImmediateReboot
  Immediate reboot.
- ExitWindows /ImmediateLogout
  Immediately stop script editing and terminate opsi-script.

Product-Properties:
For some products it's necessary to provide options. These are specifically evaluated per client at runtime. How such properties are created is described in the chapter Creation of opsi product packages
The access to the values of the properties is done via the function `GetProductProperty`:

```plaintext
if GetProductProperty("example-property", "no") = "yes"
Files_copy_extra_files
endif
```

**Encoding:**

Write your scripts in UTF-8 encoding and set the line `encoding=utf8` At the beginning of the file

### Special commands for Linux

- **GetOS** // returns: Linux or Windows_NT or MacOS [W/L/M]
- **getLinuxDistroType** // returns: debian or redhat or suse [L]
  


Here you will find the collection of all linux specific opsi-script commands:


In the following chapters special opsi Linux commands to install software are presented, which come from the opsi-script library `uib_lin_install`. This documentation was automatically generated directly from the source code.

Before we step into the documentation, some Remarks to the different approaches of the methods:

- **Distribution independent methods:**
  - `cleanupPackageSystem`
  - `installupdates`

- **Installing one or more distribution specific packages from a repository:**
  If you want to install only one package, you should use in the following methods instead of `$packagelist$` the following statement: `createStringList(<package name>)`
  The given package names have to match to the used distribution an version.
  - `debinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]
  - `redinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]
  - `suseinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]
  - `ucsinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]

- **Install or remove one or more distribution specific packages from a repository. The given package names have to match to the used distribution an version. The distribution will be detected.**
  - `genericLinInstall($packagelist$ : stringlist) : string`
  - `linuxRemoveOnePackage($packagename$ : string) : string`
  - `linuxInstallOneFile($packagefile$ : string) : string`

- **Install or check or remove one package for different distributions from a repository. The given package names have to match to the distribution specific names of the package. The distribution will be detected.**
  - `linuxInstallOneOf($packagelist$ : stringlist) : string`
  - `isOneInstalled($packagelist$ : stringlist) : string`
  - `linuxRemoveOneOf($packagelist$ : stringlist) : string`
**Documentation of opsi library:** uib_lin_install.opsiscript

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

**Documentation of local function** cleanupPackageSystem

**Definition**

cleanupPackageSystem() : void

**Description**

reads repo list und try to repair well known problems should be called after modifying the repo list or after failed installs

- Returns: nothing
- OnError: error counter increased; Error messages in the log
- Author: detlef oertel
- Date: 19.08.2020
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

**Example:**
Documentation of local function `installupdates`

**Definition**

`installupdates() : string`

**Description**

Try to install pending updates from the known repositories should be called after modifying the repo list or after failed installs.

- **Returns:** nothing
- **OnError:** Returns string "false"; error counter increased; Error messages in the log
- **Author:** detlef oertel
- **Date:** 19.08.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

**Example:**

```
10. Integration of own software into the software distribution of opsi.
```
Documentation of local function runCommandWithList

Definition

runCommandWithList($command$ : string, $list$ : stringlist) : string

Description

- Parameter: $command$
  - Type: String - Calltype: CallByValue
- Parameter: $list$
  - Type: Stringlist - Calltype: CallByValue

Author: detlef oertel
Date: 19.8.2020
Email: d.oertel@uib.de
Version: 1.0
Copyright: AGPLv3

Documentation of local function debinstall

Definition

debinstall($packagelist$ : stringlist) : string

Description

try to install the packages given by $packagelist$

- Parameter: $packagelist$
  - Type: Stringlist - Calltype: CallByValue

Returns: Returns string "true" if all is ok
OnError: Returns string "false"; error counter increased; Error messages in the log

References: [cleanupPackageSystem] [installupdates] [redinstall] [suseinstall] [ucsinstall]

Author: detlef oertel
Date: 19.08.2020
Email: d.oertel@uib.de
Version: 1.0
Copyright: AGPLv3

Example:

see: cleanupPackageSystem()
**Documentation of local function** redinstall

**Definition**

\[ \text{redinstall}(\$\text{packagelist}\$ : \text{stringlist}) : \text{string} \]

**Description**

try to install the packages given by \$\text{packagelist}\$

- **Parameter:** \$\text{packagelist}\$
  - Type: Stringlist - Calltype: CallByValue
  - Parameter \$\text{packagelist}\$ Description: stringlist with packages to install
- **Returns:** Returns string "true" if all is ok
- **OnError:** Returns string "false"; error counter increased; Error messages in the log
- **References:** [cleanupPackageSystem] [installupdates] [debinstall] [suseinstall] [ucsinstall]
- **Author:** detlef oertel
- **Date:** 19.08.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

**Example:**

see: cleanupPackageSystem()
Example:

```
see: cleanupPackageSystem()
```

---

**Documentation of local function** `ucsinstall`

**Definition**

```
ucsinstall($packagelist$ : stringlist) : string
```

**Description**

try to install the packages given by $packagelist$

- **Parameter:** `$packagelist$`
  - **Type:** Stringlist - Calltype: CallByValue
  - **Parameter $packagelist$** Description:
    stringlist with packages to install

- **Returns:** Returns string "true" if all is ok
- **OnError:** Returns string "false"; error counter increased; Error messages in the log
- **References:** [cleanupPackageSystem] [installupdates] [debinstall] [reinstall] [suseinstall]

- **Author:** detlef oertel
- **Date:** 19.08.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

---

Example:

```
see: cleanupPackageSystem()
```

---

**Documentation of local function** `genericLinInstall`

**Definition**

```
genericLinInstall($packagelist$ : stringlist) : string
```

**Description**

try to determine the Linux family and try to install the packages given by $packagelist$

- **Parameter:** `$packagelist$`
  - **Type:** Stringlist - Calltype: CallByValue
  - **Parameter $packagelist$** Description:
    stringlist with packages to install
Documentation of local function `linuxInstallOneOf`

**Definition**

\[
\text{linuxInstallOneOf}(\$\text{packagelist}\$ : \text{stringlist}) : \text{string}
\]

**Description**

Try to install any package given by \$\text{packagelist}\$ This can be used specifying a package with different names for different Linux distributions.

- **Parameter**: \$\text{packagelist}\$
  - Type: Stringlist - Calltype: CallByValue
  - Parameter \$\text{packagelist}\$ Description: stringlist with packages to install
- **Returns**: Returns string 'True' if one package was successfully installed
- **OnError**: Returns string 'False'
- **References**: [isOneInstalled] [getLinuxCommand]

**Example**:

```plaintext
[Actions]
importlib "uib_lin_install"
if isOneInstalled(createStringList("lsusb", "usbutils")) = "False"
    message "installing lsusb or usbutils"
    set $success$ = linuxInstallOneOf(createStringList("lsusb", "usbutils"))
endif
```
Documentation of local function **isOneInstalled**

**Definition**

```plaintext
isOneInstalled($packagelist$ : stringlist) : string
```

**Description**

check for installation status and return if any of $packagelist$ exists This can be used to check a package with different names for different linux distributions.

- **Parameter:** $packagelist$
  - **Type:** Stringlist - Calltype: CallByValue
  - **Description:** stringlist with packages to check
- **Returns:** Returns string 'True' if one specified package is installed
- **OnError:** Returns string 'False'
- **References:** [linuxInstallOneOf] [getLinuxCommand]
- **Author:** nils doerrer
- **Date:** 16.11.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

Documentation of local function **getLinuxCommand**

**Definition**

```plaintext
getLinuxCommand($type$ : string) : string
```

**Description**

Determine package manager and return command.

- **Parameter:** $type$
  - **Type:** String - Calltype: CallByValue
  - **Description:** type of desired command 'install','check', 'localpackage'
- **Returns:** Package manager command according to type
- **OnError:** Returns string 'False'
- **References:** [linuxInstallOneOf] [isOneInstalled]
- **Author:** nils doerrer, d.oertel
- **Date:** 14.01.2021
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3
Documentation of local function getLinuxCommandAgnostic

Definition

```
getLinuxCommandAgnostic($type : string) : string
```

Description

Determine package manager and return command.

- Parameter: `$type$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$type$` Description:
    type of desired command 'install' or 'check'
- Returns: Package manager command according to type
- OnError: Returns string 'False'
- References: [linuxInstallOneOf] [isOneInstalled] [getLinuxCommand]
- Author: nils doerrer
- Date: 16.11.2020
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Documentation of local function linuxInstallOneFile

Definition

```
linuxInstallOneFile($packagefile : string) : string
```

Description

try to install the local file package given by $packagefile$ This can be used specifying a package with different names for different linux distributions.

- Parameter: `$packagefile$`
  - Type: `String` - Calltype: `CallByValue`
- Returns: Returns string '0' if one package was successfully installed
- OnError: Returns string '-1'
- References: [isOneInstalled] [getLinuxCommand]
- Author: d.oertel
- Date: 08.02.2021
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:
Documentation of local function linuxRemoveOnePackage

Definition

\[
\text{linuxRemoveOnePackage}(\$\text{packagename}\$ : \text{string}) : \text{string}
\]

Description

try to remove the package given by $\text{packagename}$ This can be used specifying a package with different names for different linux distributions.

- Parameter: $\text{packagename}$
  - Type: \text{String} - Calltype: \text{CallByValue}
  - Parameter $\text{packagename}$ Description:
    string with the name of a package to remove

- Returns: Returns string '0' if package was successfully removed or was not installed
- OnError: Returns string '-1'
- References: [isOneInstalled] [getLinuxCommand]
- Author: d.oertel
- Date: 08.02.2021
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
[Actions]
importlib "uib_lin_install"
if stringToBool(linuxRemoveOnePackage("dummy")
  comment "success"
endif
```

Documentation of local function linuxRemoveOneOf

Definition

\[
\text{linuxRemoveOneOf}(\$\text{packagelist}\$ : \text{stringlist}) : \text{string}
\]

Description

try to remove any package given by $\text{packagelist}$ This can be used specifying a package with different names for different linux distributions.

- Parameter: $\text{packagelist}$
10. Integration of own software into the software distribution of opsi.

- Type: Stringlist - Calltype: CallByValue
- Parameter $packagelist$ Description: stringlist with packages to install
  - Returns: Returns string 'True' if one package was successfully installed
  - OnError: Returns string 'False'
- References: [isOneInstalled] [getLinuxCommand]
- Author: nils doerrer / detlef oertel
- Date: 16.11.2020
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
[Actions]
importlib "uib_lin_install"
if isOneInstalled(createStringList("lsusb", "usbutils")) = "True"
    message "installing lsusb or usbutils"
    set $success$ = linuxRemoveOneOf(createStringList("lsusb", "usbutils"))
endif
```

10.1.9. Third example: Linux-Template l-opsi-template

You can create this template with the opsi-setup-detector.
Listing 2. `define_vars_multi.opsiscript`: Variablen deklaration

```plaintext
; This is a opsi-script file.
; See https://opsi.org    https://uib.de
; This code was originally created by opsi-setup-detector 4.2.0.10

DefVar $arch$
DefVar $distCodeName$
DefVar $distRelease$
DefVar $distroName$
DefVar $distrotype$
DefVar $errorstring$
DefVar $exitcode$
DefVar $iconfile$
DefVar $installCommand$
DefVar $installSuccess$
DefVar $installdir$
DefVar $installdir1$
DefVar $installdir2$
DefVar $installerfile$
DefVar $licensekey$
DefVar $licensepool$
DefVar $licenserequired$
DefVar $minimumspace$
DefVar $msiid$
DefVar $oldProgFound$
DefVar $os$
DefVar $osshort$
DefVar $productid$
DefVar $targetfile$
DefVar $tmpstr$
DefVar $uninstallprogram$

DefStringList $ListOfPackageNames$
DefStringList $osinfomap$
```

Listing 3. `setup.opsiscript`: Installationsscript

```plaintext
; This is a opsi-script file.
; See https://opsi.org    https://uib.de
; This code was originally created by opsi-setup-detector 4.2.0.10

encoding=utf8

[Actions]
requiredOpsiScriptVersion >= "4.12.4.23"

importlib "uib_exitcode.opsiscript"
importlib "osd-lib.opsiscript"
importlib "uib_lin_install.opsiscript"

; All variables are defined here:
include_insert "define_vars_multi.opsiscript"

; Please edit the following values:
```
$ProductId$ is the name of the product in opsi, only lower letters, no umlauts, no white spaces, use '-' as a seperator

Set $ProductId$ = "l-opsi-template"
Set $MinimumSpace$ = "1 MB"

Set $LicenseRequired$ = "false"
Set $LicensePool$ = "";

Set $OS$ = GetOS
if not(($OS$ = "linux"))
   logError "Installation aborted: wrong OS version: only linux"
isFatalError "wrong OS"
endif

comment "Show product picture"
ShowBitmap "%ScriptPath%" + $ProductId$ + ".png" $ProductId$
Message "Installing " + $ProductId$ + " ..."
if $LicenseRequired$ = "true"
   comment "Licensing required, reserve license and get license key"
   set $LicenseKey$ = get_licensekey_byPool($LicensePool$)
endif

comment "Start setup"
ChangeDirectory "%SCRIPTPATH%\files1"
;----------------------------------------------
cleanupPackageSystem()
;----------------------------------------------

; To create a new repo: described in the opsi-script manual (Linux)
; install a package from an existing repo:
; set $installSuccess$ = linuxInstallOneOf($ListOfPackageNames$)
; set $exitcode$ = boolToGenericExitcode($installSuccess$)
; install a deb/rpm file:
; Belongs on the distribution. tyr to analyze with opsi-setup-detector
;----------------------------------------------
cleanupPackageSystem()
;----------------------------------------------
if "true" = isGenericExitcodeFatal($exitcode$, "true", $ErrorString$ )
   logError $ErrorString$
isFatalError $ErrorString$
else
   Comment $ErrorString$
endif

comment "Copy files"
Files_install
[Files_install]
; Example of recursively copying some files into the installation directory:
;
copy -s "%ScriptPath%\files\*."* "$InstallDir$"
;----------------------------------------------
Listing 4. uninstall.opsiscript: Deinstallations-Skript

; ------------------------------------------
; This is a opsi-script file.
; See https://opsi.org    https://uib.de
; This code was originally created by opsi-setup-detector 4.2.0.10
; ------------------------------------------
encoding=utf8

[Actions]
requiredOpsiScriptVersion >= "4.12.4.23"
importlib "uib_exitcode.opsiscript"
importlib "osd-lib.opsiscript"
importlib "uib_lin_install.opsiscript"

; All variables are defined here:
include_insert "define_vars_multi.opsiscript"

; ------------------------------------------
; Please edit the following values:
; ------------------------------------------
;$ProductId$ is the name of the product in opsi, only lower letters, no umlauts, no white spaces, use '-' as a seperator
Set $ProductId$      = "l-opsi-template"
; the path were we find the product after the installation
Set $InstallDir$    = "<none>
Set $LicenseRequired$ = "false"
Set $LicensePool$     = ""
; enter here names of the package at the supported Distributions / Versions
Set $ListOfPackageNames$ = CreateStringList("<packagename>")
; ------------------------------------------

set $OS$ = GetOS
if not(($OS$ = "linux"))
   logError "Installation aborted: wrong OS version: only linux"
   isFatalError "wrong OS"
endif

comment "Show product picture"
ShowBitmap "%ScriptPath%" + $ProductId$ + ".png" $ProductId$

Message "Uninstalling " + $ProductId$ + " ..."
if FileExists("%ScriptPath%\delsub.opsiscript")
   comment "Start uninstall sub section"
   Sub "%ScriptPath%\delsub.opsiscript"
endif

if $LicenseRequired$ = "true"
   comment "Licensing required, free license used"
   Set $tmpstr$ = FreeLicense($LicensePool$)
endif
10.2. Create an opsi product package

10.2.1. Installation of the opsi-setup-detector, opsi PackageBuilder and opsi-logviewer

Installation of the opsi-package-builder

The opsi-package-builder is currently available for Windows and Linux and MacOS.

The installation files / packages of the opsi-package-builder can be found here:
There you will find in the first lines of the post links to the installation files for Windows and Linux and MacOS.
The opsi-package-builder is not made by 'uib' but by Holger Pandel from the opsi-community (thank you!).

The opsi-package-builder open source license:
https://github.com/pandel/opsiPackageBuilder/blob/master/LICENSE_DE

The opsi-package-builder has its own documentation, that is part of the installation.

You can install the opsi-package-builder also via opsi:
The package `opsipackagebuilder_wlm` belongs to the opsi standard products and should be installed on your opsi-server. If not, use:

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

to install it on the opsi-server.

**Installation of the opsi-setup-detector**

The opsi-setup-detector is currently available for Windows and Linux and MacOS.

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:

```
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/opsi4.2/misc/helper/](https://download.uib.de/opsi4.2/misc/helper/)

The base functionality of the `opsi-setup-detector` is the same on all supported platforms. While analyzing a installer file some helper programs will be called, that may not available or runnable.

- Inno-Setups will be analyzed with `innounpack.exe` at Windows.
- wix-setups will be analyzed with `dark.exe` at Windows.
- `.deb` or `.rpm` files will be analyzed with the Linux command line tools.

The opsi product `opsi-setup-detector` has a dependency on the opsi product `opsi-package-builder_wlm`. 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.

**Installation of the opsi-logviewer**

The opsi-logviewer is currently available for Windows, Linux and MacOS.

You can install the opsi-logviewer via opsi:

The package `opsi-logviewer` is part of the opsi standard products and should be installed on your opsi-server. If not, with:

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

You can install it on the opsi-server.

A setup program to install the opsi-setup-detector on Windows even without opsi can be found at:

[https://download.uib.de/opsi4.2/misc/helper/](https://download.uib.de/opsi4.2/misc/helper/)

The opsi product `opsi-logviewer` has a stated dependency to the opsi product `javavm`.

**10.2.2. The opsi-setup-detector program to create a Linux script.**
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.

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.
Figure 8. opsi-setup-detector Start

On the main window, select the desired task and follow the dialogs or select the ‘Next step’ button.

The offered tasks are grouped by:

- OS independent
- Windows
- Linux
- MacOS
- multi platform

Die angebotenen Aufgaben für Linux:

1. **Analyze file and create opsi package**
   
   Hier wird von einer Linux Installer-Datei ausgegangen und der gesamte Ablauf bis zur Erzeugung eines opsi-Paketes durchlaufen. Dieser Prozeß ist analog dem für Windows im nächsten Kapitel beschrieben.

2. **Eine opsi Paketvorlage (Template) erzeugen**
   
   Dieser Punkt fragt nicht nach einer Setup-Datei, sondern erstellt ein opsi template Produkt für Linux bei dem die Angaben aus der Produktkonfiguration bereits übernommen werden.


**opsi-setup-detector: Analyze file and create opsi package**

The workflow is here described using *Analyze file and create 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.*
10. Integration of own software into the software distribution of opsi.

In this dialog you may choose to abort the create process. You may also choose to continue based on the pattern of a choosable installer type.

If we have a successful analyze, you will see the result page.

![Figure 11. opsi-setup-detector Result of the analysis](image)

- **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**, **deinstallation program**, and **software version** and adjust them in your script if necessary.

### opsi-setup-detector: Product configuration 1

![opsi Setup Detector](image)

**Figure 12. 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.
If the configuration contains the connection data for your opsi-server, the connection will here be started. If the configuration does not contain the password (for security reasons) you will be asked for the password:
Productid

Productid (identifier) of the product to which a dependency exists. If there is a connection to the opsi-server, this will be noticed in green letters and you may select the productid in the DropDownBox. If there is no connection, this will be noticed in red letters and you have to write the productid in the input field.

Require Mode

You can either request the Action setup or (see below) the State (installed).

Action or State

For State: State 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).

This control is disabled while creating a Meta Product to avoid sense less changes.

Requirement 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.

This control is disabled while creating a Meta Product to avoid sense less changes.

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 requiredAction: 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: Properties

Here, editable properties (product variables) can be defined for the product.
### Figure 15. opsi-setup-detector Property Editor

<table>
<thead>
<tr>
<th>Field / Function</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
</table>
| Property Name    | Name of the product variable | This identifier is displayed in the product configuration in opsi-configed and can be read within the scripts with the function `GetProductProperty`.
| Property Type    | Variable type | Possible values: `Text` / `bool` |
| Multivalue       | Determines whether the product variable can take only exactly one or multiple values | Only available for type `Text` |
| Editable         | Determines whether the default values can be overwritten with new or additional values or not | Only available for type `Text` |
| Description      | Variable function description | Displayed as tooltip in opsi-configed |
| Possible values  | Comma separated list of possible input values | If editable is set to “True”, the list can be added later within opsi-configed. Only available for type `Text` |
| Default value    | Default value | Selection list; Only available for type `text`: Free text field. Only available for type `Multivalue`: Multiple selection |

### opsi-setup-detector: Product Icon
Figure 16. opsi-setup-detector Produktkonfiguration 3 (Icon)

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
Figure 17. 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.

- 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 **Build 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 [https://forum.opsi.org/viewforum.php?f=22](https://forum.opsi.org/viewforum.php?f=22)
  - **Create opsi package** is the button that initiates the package creation. If a package with this name already exists, you will be asked if the files in the existing directory should be backuped or deleted:
More details about the opsi-setup-detector can be found in the opsi-manual:

10.2.3. The opsi PackageBuilder program to modify a script.

At the first start after the installation the opsi PackageBuilder starts in offline mode, because important configuration data for the connection with the opsi-server is missing.

![Figure 18. opsi PackageBuilder First Start: Offline Mode](image)

If the startup does not work this way and the start menu does not respond (observed under Linux / KDE), try it from the command line by specifying a path and confirm the error message that the path was not found:

```bash
opsipackagebuilder --path /home
```

Initial configuration of the opsi PackageBuilder

To enter the missing configuration data open the Settings.
In the General tab please enter the following settings:

- **configserver**: full name (FQDN) of your opsi-configserver (e.g. opsi.mycompany.org).
- **opsiadmin user**: username of a member of the group **opsiadmin** (preferably your username)
- **opsiadmin password**: the password of the user specified above. This will not be displayed and is stored encrypted. It's necessary for the opsi PackageBuilder to communicate with the opsi-server.
- **opsi Server Version**: opsi 4.1 or higher
- **opsi Workbench**: `/var/lib/opsi/workbench`
- **command execution compatibility**: opsi 4.0.4 or newer / Sudo without password
- **User**: your full name (used in changelogs)
- **Email**: your email address (used in changelogs)

In the tab Program please enter the following settings:

- **Use existing network drive**: Check the box.
- **Development folder**: Path to the directory where the opsi packages should be created. This is ideally the path to where the opsi_workbench of your opsi server is mounted.
- **script editor**: The script editor of the opsi PackageBuilder is only available for Windows unfortunately.
  - Under Windows leave it with the default settings.
- Under Linux: External editor: `/usr/local/bin/jedit`
  Command line options: (empty)

- On MacOS: External editor: `/Application/jedit`
  Command line options: (empty)

Figure 21. opsi PackageBuilder Settings: Management

In the Administration tab, we recommend the following setting, deviating from the default

- **Package**: `opsi-makepackage -v`.

Save the settings and restart the opsi PackageBuilder. The opsi PackageBuilder should now no longer report *Offline mode*.

**Install, modify and pack packages with the opsi PackageBuilder.**

Figure 22. opsi PackageBuilder Start

Use **Open package (F2)** and select the directory in which you have created with the `opsi-setup-detector` a package. (e.g.: `w:\newprod2`)

The product window opens with different tabs. The default tab is Package.

Figure 23. opsi PackageBuilder Package Tab
In this tab you see on the left side the general metadata of the opsi product as you have already been explained in Section 10.2.2.4, “opsi-setup-detector: Product configuration 1”.

On the right side you see the script files and next to it the button:

Figure 24. opsi PackageBuilder Edit button

With the button you can invoke the file in the script editor specified in the configuration and modify the script. On Windows this is the script editor of the opsi PackageBuilder.

Figure 25. opsi PackageBuilder Script editor under Windows

Key features:

- Color syntax highlighting.
- "Folding" of source code (optional: compact, with comments)
- Lexical definition customizable (to do this, the editor must be invoked via start menu entry)
- Autocomplete for syntax elements and variables
- Freely definable and reusable code blocks ("snippets")

The core component of the editor is the module Scintilla, which is also used in other well known editors, such as Notepad++. The lexical elements (syntax highlighting and folding) for the representation of the script language valid for opsi are however completely written in AutoIt, since Scintilla does not supply its own representation module for opsi scripts. Because AutoIt is an interpreter language, it's slower than other editors and is therefore only conditionally suitable for editing very large scripts, especially when source code convolution is switched on. In the settings, however, it's possible to specify whether the editor is invoke with these functions or not, provided that the call is made directly via the script tree. If the editor is open via the link in the start menu, syntax highlighting and convolution are generally switched off at startup and can be activated via the editor menu "View".

(The editor can also be open via the command line. More information about the possible command line parameters can be check with the "--help" option).
10. Integration of own software into the software distribution of opsi.

10.2.4. Testing and improving an opsi-script script

For testing and improving a script / product there are two different variants:

- Testing the created script as 'standalone' i.e. without installing it on the opsi-server and deploying it from there to the client.
- 'Integrated' testing of the complete product with installation on the server and deployment on a client.

In both cases it will be assumed here that you have created a project with the opsi-setup-detector.
'Standalone' tests

Start the application opsi-script-gui: with double click.

- Windows: Double-click the file opsi-script.exe.
  (When starting the program on a Windows 7 / 10 client, "run as administrator" must be used with the right mouse button). If the opsi-client-agent is already installed on your computer, you will find it in C:\Program files (x86)\opsi.org\opsi-client-agent\opsi-script\opsi-script.exe. If not, copy from the share \\<opsiserver\opsi_depot, from the opsi-script\windows\x86\ directory the content of this directory.

- Linux: start file /usr/bin/opsi-script.

- MacOS: Start the application /Applications/opsi-script.

You'll see the following window:

![opsi-script-gui in interactive mode](image)

**Figure 31. opsi-script-gui in interactive mode**

- With Select Script you can select the script you want to run.
- With Start you can start the script. With it, the script will be executed on this computer.
- Now open the log file with the opsi-logviewer to see how the opsi-script interprets the script. Make sure that you can adjust the displayed log level with the slider in the lower right hand corner.
- Open the script setup.opsiscript in an editor and make the desired changes (do not forget to save). There are several ways to do this:
  - Open the project in opsi PackageBuilder and open the editor from there.
  - In principle you can use any other editor of your choice.
    We recommend the editor jEdit with opsi-script syntax highlighting, that you can find in the basic opsi products.
You can now customize and save the script in the editor (you can leave the editor open). Switch to the opsi-script window and start the script again with the Start button (the script does not have to be selected again). View the log modified based on your changes in the script with the opsi-logviewer. (Don’t forget reload via context menu or toolbar button).

In this way, i.e. by repeating the points:
- Customize the script and save it
- Execute script
- Check log
you can gradually tailor your scripts to do what you need.

Hints for solving detailed problems can be found in the next chapter. The chapter after the following explains how to create an opsi product from the scripts created in this manner, which you can install on the opsi-server.

'Integrated' tests

With the 'integrated tests' the whole project is always executed by opsi on a test client. Proceed as follows:

- Open the script setup.opsiscript in an editor and make desired changes (do not forget to save). There are several ways to do this:
  - Open the project in opsi PackageBuilder and open the editor from there.
  - In principle you can also use any other editor.
  We recommend the editor jEdit with opsi-script syntax highlighting, that you can find in the basic opsi products.
- Product Packing
  - Variant 1: Open the project in the opsi PackageBuilder and start the packing via the button Pack.
  - Variant 2: Login via terminal (e.g. Putty) to the opsi-server and change to the project directory on the workbench. Pack the product with the command opsi-makepackage.
• Install the product on the opsi-server.
  - Variant 1: Start the install in the opsi PackageBuilder with the button install.
  - Variant 2: Start the install in the terminal in the project directory with the command `opsi-package-manager -i <myproctid_version.opsi>`. Where `<myproctid_version.opsi>` is the filename that was output in the previous step when packing.

• Select and start product via `opsi-configed`
  1. Select the test client in the tab Clients
  2. In the tab Product configuration select the product. If the product is not visible (which is normal after the first installation) reload the data via the menu File / Reload all data or the button on the very left of the toolbar.
  3. For the selected product set the action request setup and save.
  4. Start the client or start it via context menu on_demand if the client is running.
  5. Wait until the product has run through on the client.
     - In the tab Logfiles / instlog inspect the log file to see how the opsi-script interprets the script.
       Note that you can adjust the log level displayed here with the slider in the lower right hand corner.

• In this way, repetition of the points mentioned:
  - Adaptation of the script and saving
  - Pack product
  - Install product on the server
  - Run product on the client
  - check log
    you can gradually customize your scripts to do what you need.

10.2.5. Packing with `opsi-makepackage`

Afterwards you can pack the product. To do this, go to the root directory of the product and execute `opsi-makepackage`. Now the product will be packed.

It's recommended to create the packages immediately with an associated md5 checksum file. This file is used by `opsi-package-updater` among others to ensure package integrity after package transfer. Such a file is created automatically, but for special usage scenarios its creation can be avoided.

When transferring packages on the opsi-depotserver, ‘zsync’ can be used to transfer only differences between different packages. In order to use this method, a special `.zsync` file is needed. Such a file is created automatically, but for special usage scenarios the creation can be avoided.

If there are space problems in the temporary directory `/tmp` when creating large packages, it’s possible to specify an alternate temporary directory using `--temp-directory`.

If a package of this version already exists, `opsi-makepackage` will show a query:

```
Press <O> to overwrite, <C> to abort or <N> to specify a new version:
```

With `o` you can choose to overwrite, with `c` you cancel the process and with `n` you can choose to be asked for a new product or package version.

You can install the packed package on the server with `opsi-package-manager --install <package-file>`

More details about the `opsi-makepackage` can be found in the opsi-manual:
10.2.6. Installing with opsi-package-manager

To install the packed product there is a command `opsi-package-manager`. To do this, go to the root directory of the product and execute the following command.

```
opsi-package-manager -i <myproductid_version.opsi>
```

More details about the `opsi-package-manager` can be found in the opsi-manual:

10.2.7. Example of a 'control' file

```
[Package]
version: 1
depends:

[Product]
type: localboot
id: mytest
name: My Test
description: A test product
advice:
  version: 3.14
  priority: 10
licenseRequired: False
productClasses:
  setupScript: setup.ins
uninstallScript:
  updateScript:
  alwaysScript:
  onceScript:
  customScript:
  userLoginScript:

[ProductDependency]
action: setup
requiredProduct: javavm
requiredStatus: installed

[ProductProperty]
type: unicode
name: mytextprop
multivalue: False
editable: True
description: hint
values: ["off", "on"]
default: ["off"]

[ProductProperty]
type: bool
name: myboolprop
description: yes or no
default: False

[Changelog]
mytest (3.14-1) testing; urgency=low
  * Initial package

-- jane doe <j.doe@opsi.org>  Mi, 14 Jul 2010 12:47:53 +0000
```
10.2.8. Create opsi-package with CLI tool opsi-newprod

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

To start creating a new product, change directories to the product directory, and start the creation of the new product by entering the command `opsi-newprod`. The next question will ask you about the type of product you want to create. Choose the type `localboot` for products which should be installable by `opsi-client-agent/opsi-script`. The product type `netboot` is used for products which are activated as a bootimage (like OS installation).

Figure 33. Choose the product type: localboot

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

Figure 34. Input of the product information

**Product Id**

is a distinct short name for the product, independent from the product version (we recommend to use only plain ASCII letters and '-', no white space, no special characters)

**Product name**

is the full name of the product

**Description**

is an additional description of the product.

**Advice**

is some additional information on how to handle the product (a note).

**Product version**

is the version of the packed software (max 32 chars).
Package Version

is the version of the package for the product version. For example, this helps to distinguish between packages with the same product version but with modified opsi-script scripts.

License required

is only relevant to netboot products.

Priority

controls the installation sequence. Possible Values are between 100 (at the very beginning) and -100 (at the end). Note: product dependencies also have influence on the installation sequence. See the opsi manual for more information.

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

![Figure 35. Input of the opsi-script script names for different actions](image)

Figure 35. Input of the opsi-script script names for different actions

After editing the product information you should mention the script you want to use for different activities.

Usually the Setup script is named setup.opsiscript

Usually the Uninstall script is named uninstall.opsiscript

An Update-Script will be used for minor changes on existing big installations. If this product is switched to the required action setup, then the update script will be automatically executed after the setup script.

An Always-Script will be executed at the beginning of every activity of opsi-client-agent (e.g. on every boot).

A Once-Script has the resulting state not_installed. It is a very special kind of script, and you should only use it if you really know what you are doing.

A Custom-Script doesn't change the resulting state. It is a very special kind of script, and you should only use it if you really know what you are doing.

A userLoginScript is used to modify the user's profile after the user logs into the system. It only works with the opsi extension User Profile Management, which is described at the User Profile Management chapter in the opsi-manual.

<table>
<thead>
<tr>
<th>Type</th>
<th>resulting state</th>
<th>resulting action</th>
</tr>
</thead>
<tbody>
<tr>
<td>setup</td>
<td>installed</td>
<td>none</td>
</tr>
<tr>
<td>uninstall</td>
<td>not_installed</td>
<td>none</td>
</tr>
<tr>
<td>update</td>
<td>installed</td>
<td>none</td>
</tr>
<tr>
<td>always</td>
<td>installed</td>
<td>always</td>
</tr>
<tr>
<td>once</td>
<td>not_installed</td>
<td>none</td>
</tr>
<tr>
<td>custom</td>
<td>unchanged</td>
<td>unchangend</td>
</tr>
<tr>
<td>User login</td>
<td>unchanged</td>
<td>unchangend</td>
</tr>
</tbody>
</table>
The next step is to define one or more product dependencies. If there are no product dependencies, select No.

![Create product dependency: No/Yes](image)

**Figure 36. Create product dependency: No/Yes**

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

![Data needed to create a dependency](image)

**Figure 37. Data needed to create a dependency**

**Dependency for Action**

Which product action shall the dependency create, or when should the dependency be checked (only setup).

**Required product id**

Product id of the required product.

**Required action**

Select the required action (setup) for the required product. If no required action is set, a required installation status must be set.

**Required installation status**

Select the required status of the required product (installed). So the required product will be installed if it isn’t installed on the client yet. If no required installation status is set, a required action must be set.

**Requirement type**

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

The possibility to define uninstall actions or dependencies is broken. After defining a product dependency, you will be asked if you want to create another product dependency. If you choose Yes, then the procedure for defining a product dependency is repeated. If you choose No, then you will be asked to define some product properties, which means defining additional switches for product customization.

The installation sequence results from a combination of product dependencies and product priorities. For details on how this is done, and what you can configure, see the opsi-manual.

Figure 38. (another) product property to create?

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

The product properties are client specific, and have names (keys) which can hold different values. These values can be evaluated by the opsi-script script, and result in installing different options at installation time.

First we have to decide if our property is a text value (unicode) or a logical value e.g. true/false (boolean). If you are not sure choose unicode.

Figure 39. Choose the data type of the property

Next, a description for the switch needs to be specified. This description will be shown in the opsi-configed as a help text. Next, you can define the set of values for the switch (separated by comma). If this is left blank, then any value is allowed for the switch.

If a values contains a backslash \ it has to be doubled.
An example showing how a path would be defined: C:\\temp
Next, you can decide if the product property has a default value (switch).

If you choose `boolean` as the data type, then the description will contain only the `Property name` and `Property description`.

After defining a product property, you will be asked if you want to create another product property. If you choose `Yes`, then the procedure of defining a property will be repeated. If you choose `No`, then you will be asked for name and email of the product maintainer. This data will be written on the changelog.

Finally, the basic definitions for the new product are done.

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

11.1. The opsi directories on Linux

Wesentliche opsi Verzeichnisse und Dateien auf dem Linux-Client

- `/var/log/opsi-script` (Log files opsi-script)
- `/var/log/opsi-client-agent` (sonstige opsi Log files)
- `/usr/bin/opsi*` (Kommandozeilen Programme)
- `/opt/opsi-script`
- `/etc/opsi-client-agent` (Konfiguration opsi-client-agent)
- `/etc/opsi-script` (Konfiguration opsi-script)
- `/usr/lib` (Hilfslibraries z.B. ssl)
- `/usr/lib/opsiclientd`
- `/usr/bin/locale` (locale opsi-script)
- `/usr/share/opsi-client-agent`
- `/usr/share/opsi-script`
- `/usr/share/notifier.d`
- `/var/lib/opsi-client-agent`
12. Additional informations

The opsi Manual contains further information, which are important for the productive operation. If you use your opsi-server productively we especially recommend to get familiar with the tool 'opsi-backup' to be able to create a backup of your data.

If you don't find what you are looking for there or if you need help, please contact the opsi Community.

For productive installations we recommend professional support by uib within the framework of a maintenance and support contract.