Table of Contents

1. Copyright .................................................................................................................. 1
2. Introduction macOS clients in opsi ........................................................................... 2
   2.1. Conventions of this document ........................................................................... 2
3. Requirements for macOS Clients .............................................................................. 3
   3.1. Supported as opsi-client: MacOS ...................................................................... 3
4. Installing the minimal macOS opsi products. .............................................................. 4
   4.1. opsi standard software for macOS under opsi. ................................................. 4
   4.2. Automatic installation of the minimal macOS opsi products. ......................... 4
   4.3. Manual installation of the macOS opsi products. ........................................... 4
5. Integration of existing macOS clients into opsi. ......................................................... 7
   5.1. Using opsi-client-agent-installer on macOS .................................................... 7
   5.2. Using service_setup.sh on macOS .................................................................. 8
      5.2.1. Using service_setup.sh on macOS (initial installation). ............................ 8
      5.2.2. Using service_setup.sh on macOS (repair installation). ......................... 9
   5.3. Using opsi-deploy-client-agent for macOS. ..................................................... 10
6. Rollout of existing products to MacOS. ..................................................................... 13
   6.1. Inventory with the localboot products hwaudit and swaudit ......................... 13
   6.2. Distribution of opsi standard products: m-homebrew. ...................................... 13
   6.3. Distribution of opsi standard products: m-javavm ........................................... 13
   6.4. Distribution of opsi standard products: opsi-configed. .................................... 14
7. Integration of own software into the software distribution of opsi. .............................. 15
   7.1. A small tutorial on how to create a opsi-script script ...................................... 15
      7.1.1. Introduction ............................................................................................... 15
      7.1.2. Methods of non-interactive software installation on macOS. .................. 15
      7.1.3. Structure of an opsi-script script .............................................................. 16
      7.1.4. Primary Sections ....................................................................................... 16
      7.1.5. Important secondary sections ................................................................. 17
      7.1.6. Global constants ...................................................................................... 17
      7.1.7. Second example: tightvnc ....................................................................... 18
      7.1.8. Elementary commands for primary sections ............................................ 18
      7.1.9. Third example: macOS template m-opsi-template .................................. 26
   7.2. Create an opsi product package ........................................................................ 29
      7.2.1. Installation of the opsi-setup-detector, opsi PackageBuilder and opsi-logviewer ... 30
      7.2.2. The opsi-setup-detector program to create a MacOS script. ...................... 31
      7.2.3. The opsi PackageBuilder program to modify a script. .............................. 31
      7.2.4. Testing and improving an opsi-script script .............................................. 45
      7.2.5. Packing with opsi-makepackage ................................................................ 48
      7.2.6. Installing with opsi-package-manager ..................................................... 49
      7.2.7. Example of a 'control' file ...................................................................... 49
      7.2.8. Create opsi-package with CLI tool opsi-newprod .................................... 50
8. General notes about MacOS. ...................................................................................... 55
   8.1. The Keyboard under MacOS ............................................................................. 55
   8.2. The opsi product m-homebrew ....................................................................... 56
   8.3. The MacOS pseudo user opsisetupadmin ....................................................... 57
   8.4. The opsi directories on MacOS ....................................................................... 57
   8.5. MacOS directory structure with ‘go’ and ‘nogo’. ............................................ 58
9. Additional informations ............................................................................................ 59
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 macOS clients in opsi

This manual describes the operation of macOS 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. Adding and integrating macOS computers in opsi (Installation of the opsi-mac-client-agent)
2. Deployment of opsi standard software for macOS on the opsi-server
3. Installation of standard software on macOS clients
4. opsi standard software for macOS under opsi
5. Packaging of own software
6. Creation of opsi packages
7. Notes about macOS clients
   a. Special commands for macOS
   b. Directories you may use
   c. The pseudo user opsisetupadmin

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 macOS Clients

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


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).

Further details can be found in


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

Supported macOS versions:

### 3.1. Supported as opsi-client: MacOS

As of 16.02.2022

<table>
<thead>
<tr>
<th>Distribution</th>
<th>client-agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacOS 10.13 HighSierra</td>
<td>🔴</td>
</tr>
<tr>
<td>MacOS 10.14 Mojave</td>
<td>🔴</td>
</tr>
<tr>
<td>MacOS 10.15 Catalina</td>
<td>✔️</td>
</tr>
<tr>
<td>MacOS 11 BigSur</td>
<td>✔️</td>
</tr>
<tr>
<td>MacOS 12 Monterey</td>
<td>✔️</td>
</tr>
</tbody>
</table>

✔️: Supported ✗: Unsupported 🔴: Under Development 🔴: Discontinued

When using the arm64 architecture (Apple Silicon, M1), it's currently necessary to install the opsi-client-agent of the 'Dynamic Binary Translators' called *rosetta2*. You can install this with:

```
softwareupdate --install-rosetta --agree-to-license
```

You can verify the successful installation with:

```
pkgutil --pkgs | grep Rosetta

```

Native support for the arm64 architecture (Apple silicon) is planned. The necessary reference devices have been ordered for this purpose. To disclose when the deployment for this platform will take place cannot be made at this time (April 2022).
4. Installing the minimal macOS opsi products.

For the distribution of software with opsi ready products are available for installation. These contain among other things the agent ('opsi-client-agent'), which must be installed for the management on clients.

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

4.1. opsi standard software for macOS under opsi.

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

- opsi-mac-client-agent
- swaudit
- hwaudit
- m-homebrew (also check: the opsi product m-homebrew)
- m-system-update
- opsi-configed
- opsi-logviewer
- opsi-auto-update
- m-javavm
- opsi-setup-detector
- windomain

4.2. Automatic installation of the minimal macOS opsi products.

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

The configuration of the opsi repositories for macOS clients can be found in the directory /etc/opsi/package-updater.repos.d/ in the files uib-mac-testing.repo and uib-mac.repo.

Activate the desired repos by setting the entry active = true in the chosen *.repo file.
4. Installing the minimal macOS opsi products.

Listing 1. `/etc/opsi/package-updater.repos.d/uib-mac-testing.repo`

```ini
[repository_uib_macos_testing]
description = opsi macos packages in testing
active = true
baseUrl = http://download.uib.de
dirs = opsi4.2/testing/packages/macos/localboot/
autoInstall = false
autoUpdate = true
autoSetup = false
; Set Proxy handler like: http://10.10.10.1:8080
proxy =
```

Listing 2. `/etc/opsi/package-updater.repos.d/uib-mac.repo`

```ini
[repository_uib_macos_stable]
description = opsi macos packages in stable
active = true
baseUrl = http://download.uib.de
dirs = opsi4.2/stable/packages/macos/localboot/
autoInstall = false
autoUpdate = true
autoSetup = false
; Set Proxy handler like: http://10.10.10.1:8080
proxy =
```

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

```bash
opsi-package-updater -v --repo uib_macos_stable install
```

or

```bash
opsi-package-updater -v --repo uib_macos_testing install
```

After a successful installation you have to reload all data on the opsi-configed to make the new products become visible there.

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

```
[repository_uib_macos_stable]
...
proxy =
```

Should installed packages be updated later, this can be done with the following command:

```bash
opsi-package-updater -v update
```

It is also possible to manually download and install the packages.

Get the latest opsi packages in the .opsi package format. You can find the packages at https://download.uib.de/opsi4.2/stable/packages/macos/localboot or at https://download.uib.de/opsi4.2/testing/packages/macos/localboot.

We recommend saving the .opsi files to /var/lib/opsi/repository. To ensure that opsiconfd can access the files, opsi-set-rights /var/lib/opsi/repository should be run.

After downloading, you need to install the packages on the server using the command opsi-package-manager -i <package-name>.opsi.
5. Integration of existing macOS clients into opsi.

To include existing macOS 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, it 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 macOS 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.

5.1. Using opsi-client-agent-installer on macOS

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
3. Execute the installer (for linux and macos this must be done with root-rights, on windows a UAC-Request may be displayed)
4. 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 a old opsiclient.conf is found), but you may need to add or change some of the data.

- Client-ID should be the fqdn of the Client.
5. Integration of existing macOS clients into opsi.

- 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.


5.2. Using service_setup.sh on macOS

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.

5.2.1. Using service_setup.sh on macOS (initial installation).

Due to macOS security restrictions, the ability to run scripts from mounted shares is limited. Trying to do the following with a share mounted via Finder to /Volumes (or similar) will therefore fail (depending on the macOS version).

- 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 `<servicepass>` together with the mount option `-N`, then you will be prompt to input the password
  - `<opsi-server>` the name or IP number of the opsi-server.
5. Integration of existing macOS clients into opsi.

5.2.1. Using service_setup.sh on macOS (initial installation).

1. Start from the mounted share the script `opsi-mac-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 reboots after the installation.

5.2.2. Using service_setup.sh on macOS (repair installation).

- Log in to the client.
- Start the terminal program
- For the following commands you need to replace the following placeholders:
  - `<serviceuser>` with a username known on the opsi-server.
  - `<servicepass>` with the password of the `<serviceuser>`. You can also omit `:<servicepass>`, then you will be asked for the password
  - `<opsi-server>` with the name or IP number of the opsi server.
- During the first installation opsi created a hidden pseudo user named `opsisetupadmin`, in whose home directory...
5. Integration of existing macOS clients into opsi.

`/var/opsisetupadmin` is also the mount directory.

```bash
sudo su
mount_smbfs -N //<serviceuser>:<servicepass>@<opsi-server>/opsi_depot /var/opsisetupadmin/opsi_depot
cd /var/opsisetupadmin/opsi_depot/opsi-mac-client-agent
./service_setup.sh
cd
umount /var/opsisetupadmin/opsi_depot
```

Example:

```bash
sudo su
mount_smbfs -N //adminuser:linux123@sepia/opsi_depot /var/opsisetupadmin/opsi_depot
cd /var/opsisetupadmin/opsi_depot/opsi-mac-client-agent
./service_setup.sh
cd
umount /var/opsisetupadmin/opsi_depot
```

1. Start the script `opsi-mac-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 the 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 the corresponding service URL (opsi-configserver), user name and password. Here a user is needed which is a member of the group `opsiadmin`. It is also possible to utilize a user which is only allowed to execute the method `host_createOpsiClient`.

   The client needs a reboot after the installation to become active.
   The reboot is **not** triggered automatically.

### 5.3. Using opsi-deploy-client-agent for macOS.

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

Unfortunately on macOS the ssh accesses are deactivated by default. To use the `opsi-deploy-client-agent` command these accesses must be first activated.

This can be performed interactively in the 'System preferences / sharing':
On the command line, this can be done as follows:

```bash
sudo launchctl load -w /System/Library/LaunchDaemons/ssh.plist
```

A checkup of the current status of the ssh access is possible with the command:

```bash
sudo systemsetup -getremotelogin
```

Disabling ssh access on the command line looks like this:

```bash
sudo launchctl unload /System/Library/LaunchDaemons/ssh.plist
```

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-mac-client-agent
```

Typical calls are:

- For a single client:

```bash
./opsi-deploy-client-agent -v --user=uib uib-mmini1
```

Results in the following output:
### 5. Integration of existing macOS clients into opsi.

Password is required for deployment.

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:43:43.121</td>
<td>Starting deployment to host uib-mmini1.uib.local (posix.py:84)</td>
</tr>
<tr>
<td>16:43:43.121</td>
<td>Querying for ip address of host uib-mmini1.uib.local (common.py:158)</td>
</tr>
<tr>
<td>16:43:44.166</td>
<td>Got ip address 192.168.10.70 from syscall (common.py:167)</td>
</tr>
<tr>
<td>16:43:44.146</td>
<td>Patching config.ini (posix.py:91)</td>
</tr>
<tr>
<td>16:43:44.157</td>
<td>Copying installation scripts... (posix.py:107)</td>
</tr>
<tr>
<td>16:43:48.316</td>
<td>Running installation script... (posix.py:147)</td>
</tr>
<tr>
<td>16:43:55.620</td>
<td>1/1 deployments successfully (<strong>init</strong>.py:210)</td>
</tr>
</tbody>
</table>

- For a list of clients:

  ```
  ./opsi-deploy-client-agent -v --user=uib --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.

- Display all command line parameters:

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

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.

6.1. Inventory with the localboot products hwaudit and swaudit.

In opsi-script, '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.

6.2. Distribution of opsi standard products: m-homebrew.

This product installs the package management program homebrew which is used by several other opsi products for MacOS, e.g. to install Java.

See also: The opsi product m-homebrew.

In opsi-script, 'Client configuration' mode, select the client in question under the 'Clients' tab.

Switch to the 'Product Configuration' tab, click in the 'Requested' column for the m-homebrew product, this will open a list/dropdown menu and there select the 'setup' action.

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 restart the client or push the installation via 'on_demand'. It should now start the opsi-client-agent and install the m-homebrew product.

6.3. Distribution of opsi standard products: m-javavm

This product installs the Java Runtime Environment which is used by several other opsi products for MacOS, e.g. opsi-configed, opsi-logviewer.

In opsi-script, mode 'Client configuration', select the client under the tab 'Clients'.

Switch to the 'Product Configuration' tab, click in the 'Requested' column for the m-javavm product, this will open a list/dropdown menu and there please select the 'setup' action.

Attention: m-homebrew and m-javavm must be already installed!

To the standard products belongs the product opsi-configed which installs the opsi management interface as application on a computer. Because this application is a Java application, a JavaRE is included.

In the opsi-script, mode 'Client configuration', under the tab 'Clients' select the concerning client.

If not already done, update the opsi-configed's dataset using 'Reload File/Data' by clicking the corresponding icon.

Switch to the 'Product configuration' tab, click in the 'Requested' column for the opsi-configed product, this will open a list/dropdown menu and there select the 'setup' action.

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 opsi-configed product. After the installation is finished you should find the item opsi-configed under Applications.
7. 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.

7.1. A small tutorial on how to create a opsi-script script.

7.1.1. 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

7.1.2. Methods of non-interactive software installation on macOS.

Apple (unlike Microsoft) standardized its software installation methods very early on. In essence, there are two methods:

- application Directory:
  This is a directory accordinng to the pattern: <application name>.app However, such a directory is not displayed in the Finder as a directory, but as an ‘application’. Inside this directory, according to a pattern, the files of the application must be located. A typical structure would be:

```plaintext
opsi-script.app
  |--Contents
      |--Info.plist
      |--PkgInfo
      |--MacOS
          |--opsi-script
          |--opsi-script-gui
          |
      |--Resources
          |-- (...)
```
Such a directory must only be copied into the path `/Applications` for installation. Possible files in the directory MacOS must be made executable.

Such `.app` directories are usually offered packed for download.

- **PKG file:**
  These files contain software which need to be installed by a special command.

In both cases an *unattended* (i.e. non-interactive) installation is not an issue.

Often MacOS software is offered in packed formats like `*.zip`, `*.dmg` or also `*.tgz`.

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

### 7.1.3. Structure of an opsi-script script

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

First an example for a simple opsi-script script:

```
[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`.

### 7.1.4. 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
7.1.5. 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.

7.1.6. 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).
Examples:

%ProgramFiles32Dir%
c:\Program Files (x86)

%Systemroot%
c:\windows

%System%
c:\windows\system32

%opsiTmpDir%
c:\

%Scriptpath%
<path to running script>

7.1.7. 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")
```

7.1.8. 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 macOS**

- `GetOS` // returns: Linux or Windows_NT or macos [W/L/M]
- `getMacosVersionInfo` [M]
- `getMacosVersionMap` [M]


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

**Documentation of opsi library: uib_macosinstalllib.opsiscript**

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

**Documentation of local function** `install_macos_app`

**Definition**

```plaintext
install_macos_app($myapp$ : string) : string
```

**Description**

try to install the app given by $myapp$

- **Parameter:** $myapp$
  - Type: `String` - Calltype: `CallByValue`
  - Parameter $myapp$ Description: string with path to app to install
- **Returns:** Returns string "true" if all is ok
- **OnError:** Returns string "false"; error counter increased; Error messages in the log
- **References:** `install_macos_pkg` `install_macos_dmg` `install_macos_zip` `install_macos_generic`
- **Author:** detlef oertel
- **Date:** 28.12.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
Example:

```plaintext
[Actions]
importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.app (directory) and install with install_macos_app"
set $installfile$ = "%scriptpath%/files/my_macos_app.app"
set $installresult$ = install_macos_app($installfile$)
if stringtobool($installresult$)
    comment "installation succeeded"
else
    LogError "installation failed"
endif
```

Documentation of local function `install_macos_pkg`

**Definition**

`install_macos_pkg($mypkg$ : string) : string`

**Description**

try to install the pkg file given by $mypkg$

- **Parameter:** $mypkg$
  - Type: String - Calltype: CallByValue
  - Parameter $mypkg$ Description: string with path to pkg file to install
- **Returns:** Returns string "true" if all is ok
- **OnError:** Returns string "false"; error counter increased; Error messages in the log
- **References:** [install_macos_app] [install_macos_dmg] [install_macos_zip] [install_macos_generic]
- **Author:** detlef_oertel
- **Date:** 28.12.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

Example:
importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.pkg and install with install_macos_pkg"
set $installfile$ = "$scriptpath%/files/my_macos_app.pkg"
set $installresult$ = install_macos_pkg($installfile$)
if stringtobool($installresult$)
    comment "installation succeeded"
else
    LogError "installation failed"
endif

**Documentation of local function** install_macos_dmg

**Definition**

```
install_macos_dmg($mydmg$ : string) : string
```

**Description**

try to install the dmg file given by $mydmg$

- **Parameter:** $mydmg$
  - Type: **String** - Calltype: **CallByValue**
  - Parameter $mydmg$ Description:
    string with path to dmg file to install
- **Returns:** Returns string "true" if all is ok
- **OnError:** Returns string "false"; error counter increased; Error messages in the log
- **References:** [install_macos_app] [install_macos_pkg] [install_macos_zip] [install_macos_generic]
- **Author:** detlef oertel
- **Date:** 28.12.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

**Example:**
importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.dmg and install with install_macos_dmg"
set $installfile$ = "%scriptpath%/files/my_macos_app.dmg"
set $installresult$ = install_macos_dmg($installfile$
if stringtobool($installresult$)
    comment "installation succeeded"
else
    LogError "installation failed"
endif

Documentation of local function install_macos_zip

Definition

install_macos_zip($myzip$ : string) : string

Description

try to install the zip file given by $myzip$ unzips the file and try to find a installable part (*.app, *.pkg, *.dmg) and try to install this

• Parameter: $myzip$
  - Type: String - Calltype: CallByValue
  - Parameter $myzip$ Description: string with path to zip file to install

• Returns: Returns string "true" if all is ok

• OnError: Returns string "false"; error counter increased ; Error messages in the log

• References: [install_macos_app] [install_macos_dmg] [install_macos_pkg] [install_macos_generic]

• Author: detlef oertel

• Date: 28.12.2020

• Email: d.oertel@uib.de

• Version: 1.0

• Copyright: AGPLv3

Example:
[Actions]
importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.zip and install with install_macos_zip"
set $installfile$ = "%scriptpath%/files/my_macos_app.zip"
set $installresult$ = install_macos_zip($installfile$)
if stringtobool($installresult$)
    comment "installation succeeded"
else
    LogError "installation failed"
endif

Documentation of local function install_macos_generic

Definition

install_macos_generic($myfile$ : string) : string

Description

try to install the file given by $myfile$ Checks if the file is a well known installable (*.app, *.pkg, *.dmg, *.zip) and try to install this

• Parameter: $myfile$
  • Type: String - Calltype: CallByValue
  • Parameter $myfile$ Description:
    string with path to pkg file to install
• Returns: Returns string "true" if all is ok
• OnError: Returns string "false"; error counter increased ; Error messages in the log
• References: [install_macos_app] [install_macos_dmg] [install_macos_zip] [install_macos_pkg]
• Author: detlef oertel
• Date: 28.12.2020
• Email: d.oertel@uib.de
• Version: 1.0
• Copyright: AGPLv3

Example:
see: install_macos_generic

importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.* and install with install_macos_generic"
set $installfile$ = "%scriptpath%/files/opsi-script.app"
set $installresult$ = install_macos_generic($installfile$)
if stringtobool($installresult$)
    comment "installation succeeded"
else
    Error "installation failed"
endif

7.1.9. Third example: macOS template m-opsi-template.

You can create this template with the opsi-setup-detector.

Listing 3. define_vars_multi.opsiscript: Variablen deklaration

```
; 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
;-------------------------------------
; include file for opsi-setup-detector products
; Define all variables here
;-------------------------------------
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 4. setup.opsiscript: Installationsscript

; This is an 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_macosinstalllib.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 separator
Set $ProductId$ = "m-opsi-template"
Set $MinimumSpace$ = "1 MB"
; the path were we find the product after the installation
; Set $InstallDir$ = "/Applications/<product.app>"
Set $InstallDir$ = "/Applications/<productid.app>"
Set $LicenseRequired$ = "false"
Set $LicensePool$ = ""

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

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

if FileExists("%ScriptPath%/delsub.opsiscript")
  comment "Start uninstall sub section"
  Sub "%ScriptPath%/delsub.opsiscript"
endif

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"

;----------------------------------------------
set $installSuccess$ = install_macos_generic("%SCRIPTPATH%/files1/<my macos install file>")
;----------------------------------------------
if $installSuccess$ = "false"
  logError "Installation failed"
  isFatalError "Installation failed"
else
  Comment "Installation success"
endif
Listing 5. delsub.opsiscript: Ausgelagerte Deinstallations-Sub-Sektion

```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

Message "Check for existing installation of " + $ProductId$ + " ..."

comment "Start the Uninstall check:"
set $oldProgFound$ = "false"
if directoryExists($InstallDir$)
    set $oldProgFound$ = "true"
endif
if $oldProgFound$ = "true"
    Message "Uninstalling " + $ProductId$ + " ..."
    if not((InstallDir$ = '') or (InstallDir$ = 'unknown') or (InstallDir$ = '/Applications/'))
        Files_uninstall
    endif
endif

[Files_uninstall]
del -sf "$InstallDir$"/
```

7. Integration of own software into the software distribution of opsi.
7. Integration of own software into the software distribution of opsi.

Listing 6. uninstall.opsiscript: Deinstallations-Skript

; This is an 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_macosinstalllib.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 $ProductId$ = "m-opsi-template"
; the path were we find the product after the installation
Set $InstallDir$ = "'/Applications/<productid.app>"
Set $LicenseRequired$ = "false"
Set $LicensePool$ = ""

; ----------------------------------------------------------------
set $OS$ = GetOS
if not(($OS$ = "macos"))
    logError "Installation aborted: wrong OS version: only macos"
    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

7.2. Create an opsi product package
7. Integration of own software into the software distribution of opsi.

7.2.1. 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: [link](https://forum.opsi.org/viewtopic.php?p=32473#p32473)

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: [link](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: [link](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:

```bash
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/

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

### 7.2.2. The opsi-setup-detector program to create a MacOS 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.

![Opsi-setup-detector](image)

**Figure 2. opsi-setup-detector Necessary configuration on first startup**

After the first start of the opsi-setup-detector a configuration mask appears. The following information is required here:

- **fullname**: (Used for entries in the changelog.txt)
- **email_address**: (Used for entries in the changelog.txt)
- **workbench_path**: Path to the directory in which the opsi packages are going to be created. This is ideally the path to the place
where the opsi_workbench of your opsi-server is mounted.

After all needed configurations has be done and saved, you will see the startpage.

![Figure 3. opsi-setup-detector Start](image)

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

The offered tasks for macOS:

1. **Analyze file and create opsi package**
   Here it starts from a macOS installer file and then goes through the whole process to create an opsi package. This process is like it is described for Windows in the next chapter.

2. **Create an opsi package template**
   This point does not ask for a installer file, but creates a opsi template product for macOS with the information from the product configuration is already taken over.

The following screenshots show the use of Windows installer files, but they look similar when using MacOS installer files like *.app, *.dmg, *.zip.

**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 analysis will start directly after the selection is done.

**opsi-setup-detector: Analyze**

If the analysis found no result, you will get here a *Sorry unknown Installer*. 
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 6. opsi-setup-detector Result of the analysis**

- **Detected Setup Type**: Type of detected Installer
- **MST allowed**:
- **Link** with information about the installer
- **Setup file**: Path and name of the analyzed setup file
- **MST file**: For MSI installers or installers which contain MSI, an MST file can be specified here which will be integrated into the MSI call.
- **MsId**: For MSI installers or installers that contain MSI in the form of product code
- **Software version**: The version of the software to be installed if determinable.
- **Setup file size MB**: Size of the setup file in MB
- **Required space MB**: This value is an estimate of six times the size of the setup file and can be adjusted if necessary
- **InstallDir**: As far as detected the directory where the software will be installed.
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

![Figure 7. opsi-setup-detector Product configuration 1](image)

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

**opsi-setup-detector: Product 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**
• 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

• 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:

7.2.3. 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 13. opsi PackageBuilder  First Start: Offline Mode

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:

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 **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.
7. Integration of own software into the software distribution of opsi.

- Under Linux: External editor: /usr/local/bin/jedit
  Command line options: (empty)
- On MacOS: External editor: /Applications/jedit
  Command line options: (empty)

Figure 16. 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 17. 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 18. 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 opsi-setup-detector: Product configuration 1.

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

![Figure 19. opsi PackageBuilder Edit button](image)

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 20. opsi PackageBuilder Script editor under Windows](image)

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).
Figure 21. opsi PackageBuilder Product variables tab (Properties)

In this tab you see on the left side the product properties of the opsi product like they are already explained in opsi-setup-detector: Properties.

Figure 22. opsi PackageBuilder Dependencies tab

In this tab you can see on the left side the product dependencies of the opsi product like they are already explained in opsi-setup-detector: Priority and dependencies.

Figure 23. opsi PackageBuilder Button: Pack

This button starts an SSH connection from the server and executes the packaging command there. You can also do the same in a terminal itself as described in Packing with opsi-makepackage.

Figure 24. opsi PackageBuilder Button: Install

This button starts an SSH connection from the server and executes the installation command there to install the product on the server. You can also do the same in a terminal itself as described in Installing with opsi-package-manager.

Figure 25. opsi PackageBuilder Button: Installieren + Setup

Do not use it!

7.2.4. 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)

- 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.
7. Integration of own software into the software distribution of opsi.

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**.
7. Integration of own software into the software distribution of opsi.

- 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.

7.2.5. 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>`.

7.2.6. 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>
```


7.2.7. 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
```
7.2.8. 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).

![Picture of choosing product type](image)

**Figure 28. 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.

![Picture of inputting product information](image)

**Figure 29. 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:

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

**Figure 30. 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>unchanged</td>
</tr>
<tr>
<td>User login</td>
<td>unchanged</td>
<td>unchanged</td>
</tr>
</tbody>
</table>
The next step is to define one or more product dependencies. If there are no product dependencies, select No.

**Figure 31. Create product dependency: No/Yes**

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

**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 33. A(nother) product property to create?

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

The product properties are client specific, and 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 34. 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.
# 8. General notes about MacOS

## 8.1. The Keyboard under MacOS

### Table 2. Apple Tasten

<table>
<thead>
<tr>
<th>Key combination</th>
<th>Designation</th>
<th>Key combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>⌘</td>
<td>Command key</td>
<td>&quot;command&quot;</td>
</tr>
<tr>
<td>⌥</td>
<td>Option key</td>
<td>&quot;option&quot;</td>
</tr>
<tr>
<td>⇧</td>
<td>Shift key</td>
<td>&quot;shift&quot;</td>
</tr>
<tr>
<td>CMD</td>
<td>Shift key</td>
<td>&quot;shift&quot;</td>
</tr>
<tr>
<td>CTRL</td>
<td>Shift key</td>
<td>&quot;shift&quot;</td>
</tr>
</tbody>
</table>

### Table 3. Sonderzeichen

<table>
<thead>
<tr>
<th>Special character</th>
<th>Designation</th>
<th>Key combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>At</td>
<td>cmd + L</td>
</tr>
<tr>
<td>€</td>
<td>Euro</td>
<td>alt + E</td>
</tr>
<tr>
<td>$</td>
<td>Dollar</td>
<td>shift + 4</td>
</tr>
<tr>
<td>£</td>
<td>British Pound</td>
<td>alt + shift + 4</td>
</tr>
<tr>
<td>©</td>
<td>Copyright</td>
<td>alt + G</td>
</tr>
<tr>
<td>®</td>
<td>Registered</td>
<td>alt + R</td>
</tr>
<tr>
<td>™</td>
<td>Trademark</td>
<td>alt + shift + D</td>
</tr>
<tr>
<td>?</td>
<td>Apple</td>
<td>alt + shift + +</td>
</tr>
<tr>
<td>§</td>
<td>Paragraph</td>
<td>shift + 3</td>
</tr>
<tr>
<td>`</td>
<td>Tilde</td>
<td>alt + N (dann Leertaste)</td>
</tr>
<tr>
<td>(</td>
<td>opening Bracket</td>
<td>shift + 8</td>
</tr>
<tr>
<td>)</td>
<td>closing Bracket</td>
<td>shift + 9</td>
</tr>
<tr>
<td>[</td>
<td>opening Square bracket</td>
<td>alt + 5</td>
</tr>
<tr>
<td>]</td>
<td>closing Square bracket</td>
<td>alt + 6</td>
</tr>
<tr>
<td>{</td>
<td>opening Curly bracket</td>
<td>alt + 8</td>
</tr>
<tr>
<td>}</td>
<td>closing Curly bracket</td>
<td>alt + 9</td>
</tr>
<tr>
<td>/</td>
<td>Slash</td>
<td>shift + 7</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
<td>alt + shift + 7</td>
</tr>
<tr>
<td>alt + 7</td>
<td>»</td>
<td>Right double quote</td>
</tr>
<tr>
<td>alt + ^</td>
<td>&quot;</td>
<td>Left double quote</td>
</tr>
<tr>
<td>alt + shift + ^</td>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td>alt + S</td>
<td>`</td>
<td>Single quote</td>
</tr>
<tr>
<td>alt + #</td>
<td>»</td>
<td>Double right angle quotation mark</td>
</tr>
<tr>
<td>alt + shift + Q</td>
<td>«</td>
<td>Double left angle quotation mark</td>
</tr>
<tr>
<td>alt + Q</td>
<td>›</td>
<td>Single right angle quotation mark</td>
</tr>
<tr>
<td>alt + shift + N</td>
<td>‹</td>
<td>Single left angle quotation mark</td>
</tr>
</tbody>
</table>
8.2. The opsi product m-homebrew

There are a number of package management programs for MacOS, which allow to provide common Unix programs under MacOS. Typical examples of this package management programs are:

- 'homebrew' ([https://brew.sh/](https://brew.sh/))
- 'macport' ([https://www.macports.org/](https://www.macports.org/))
- 'fink' ([https://finkproject.org/](https://finkproject.org/))

The opsi product **m-homebrew** installs the **homebrew** system on a MacOS system.

**homebrew** installs the command **brew** on the system, which allows then programs to be downloaded from the Internet. Either a binary is installed directly or the sources are downloaded and the program is compiled on the machine. Therefore **homebrew** also needs the development command line tools to compile programs (**xcode-select --install**).

For security reasons **homebrew** is conceived to **not be executed as root**, but only with user rights. To ensure this in the opsi context a hidden user named **opsisetupadmin** is created during the installation of the **opsi-client-agent** (see also The MacOS pseudo user **opsisetupadmin**).

This user owns the whole **homebrew** system on the machine and therefore, only this user should be used to perform the installation of software on the machine via the **brew** command.

I.e. The **brew** command should be executed as root as follows:

```
sudo -H -u opsisetupadmin /usr/local/bin/brew
```

For example, the installation of Java in the product **m-javavm** is implemented as follows:
8. General notes about MacOS

8.3. The MacOS pseudo user opsisetupadmin

When installing the opsi-client-agent a 'hidden user' named opsisetupadmin is created. This user has no password and cannot be used for login.

The home directory of this user is /var/lib/opsisetupadmin.

This user is employed for the following purposes:

- Operating the homebrew system (see The opsi product m-homebrew).
- The directory /var/lib/opsisetupadmin/opsi_depot is the mount point on which the opsiclientd mounts the depotshare. This is done since access to network shares not mounted in a user directory are very limited.

8.4. The opsi directories on MacOS

Essential opsi directories and files on the MacOS client

- /var/log/opsi-script (Log files opsi-script)
- /var/log/opsi-client-agent (other opsi Log files)
- /usr/local/bin/opsi* (Command line programs)
- /Applications/opsi-script-gui.app
- /Applications/opsi-notifier.app
- /etc/opsi-client-agent (Configuration opsi-client-agent)
- /etc/opsi-script (Configuration opsi-script)
- /usr/local/lib (Help libraries e.g. ssl)
- /usr/local/lib/opsiclientd
- /usr/local/bin/locale (local opsi-script)
- /usr/local/share/opsi-client-agent
- /usr/local/share/opsi-script
- /usr/local/share/notifier.d
8. General notes about MacOS

8.5. MacOS directory structure with 'go' and 'nogo'.

MacOS is based on BSD-Unix and has a typical Unix file system structure. However, due to security reasons, there are several areas that are exclusively reserved for Apple and a few other specifics that are important for the person doing the packaging work:

- **/Applications**
  This is the directory where the graphical applications are located. An application consists of a directory `<Application name>.app`. Such a directory is not shown in the Finder as a directory, but as an 'Application'. Inside this directory the files of the application must be located according to a certain specific pattern.

- **/sbin, /bin, /usr/bin, /usr/share, /usr/lib**
  These are apple only directories into which nobody else is allowed to copy anything. Use instead the local variants: `/usr/local/bin, /usr/local/share, /usr/local/lib`
9. 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.