## Table of Contents

1. Copyright .................................................................................................................. 1
2. Introduction MacOS Clients in opsi ............................................................................. 2
3. Conventions of this document .................................................................................... 3
4. Requirements for MacOS Clients ................................................................................ 4
5. Installing the minimal MacOS opsi products. .............................................................. 5
   5.1. opsi standard software for macOS under opsi. ...................................................... 5
   5.2. Automatic installation of the minimal MacOS opsi products. ............................... 5
   5.3. Manual installation of the MacOS opsi products. ............................................... 6
6. Integration of existing MacOS clients into opsi. .......................................................... 7
   6.1. Using service_setup.sh on macOS ........................................................................ 7
   6.1.1. Using service_setup.sh on macOS (initial installation). ................................. 7
   6.1.2. Using service_setup.sh on macOS (repair installation). .................................. 8
   6.2. Using opsi-deploy-client-agent for macOS........................................................... 9
7. Rollout of existing products to MacOS. ...................................................................... 11
   7.1. Inventory with the localboot products hwaudit and swaudit. ............................... 11
   7.2. Distribution of opsi standard products: m-homebrew. .......................................... 11
   7.3. Distribution of opsi standard products: m-javavm ................................................. 11
   7.4. Distribution of opsi standard products: opsi-configured ....................................... 12
8. Integration of own software into the software distribution of opsi. ............................ 13
   8.1. A small tutorial on how to create an opsi-script script. ........................................ 13
       8.1.1. Introduction .................................................................................................... 13
       8.1.2. Methods of non-interactive software installation on macOS. ......................... 13
       8.1.3. Structure of an opsi-script script .................................................................. 14
       8.1.4. Primary Sections ......................................................................................... 14
       8.1.5. Important secondary sections ...................................................................... 15
       8.1.6. Global constants ........................................................................................ 15
       8.1.7. Second example:.tightvnc ........................................................................... 16
       8.1.8. Elementary commands for primary sections ................................................ 16
       8.1.9. Third example: MacOS template 'm-opsi-template'. .................................... 22
   8.2. Create an opsi product package ............................................................................ 25
       8.2.1. Installation of the opsi-setup-detector, opsi PackageBuilder and opsi-logviewer 26
       8.2.2. The opsi-setup-detector program to create a MacOS script. ............................. 27
       8.2.3. The opsi PackageBuilder program to modify a script. ................................... 35
       8.2.4. Testing and improving an opsi-script script .................................................... 39
       8.2.5. Packing with opsi-makepackage .................................................................. 42
       8.2.6. Installing with opsi-package-manager ............................................................ 43
       8.2.7. Example of a 'control' file ........................................................................... 43
9. General notes about MacOS ....................................................................................... 44
   9.1. The Keyboard under MacOS .................................................................................. 44
   9.2. The opsi product m-homebrew ............................................................................ 45
   9.3. The MacOS pseudo user opsisetupadmin ............................................................. 46
   9.4. The opsi directories on MacOS ............................................................................ 46
   9.5. MacOS directory structure with 'go' and 'nogo'. .................................................. 46
10. Additional informations ............................................................................................ 48
Chapter 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.
Chapter 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 Mac computers in opsi (Installation of the opsi-mac-client-agent)
2. Deployment of opsi standard software for Mac on the opsi-server
3. Installation of standard software on Mac clients
4. opsi standard software for Mac under opsi
5. Packaging of own software
6. Creation of opsi packages
7. Notes about Mac clients
   a. Special commands for MacOS
   b. Directories you may use
   c. The pseudo user opsisetupadmin
Chapter 3. 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.

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
Chapter 4. Requirements for MacOS Clients

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

The opsi-mac-client-agent is a co-funded 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 (email to info@uib.de).


Technical requirements is an opsi-server with opsi 4.1.

The supported Mac-Clients are:

- MacOS versions from 10.13 to 11.0 (High Sierra to Big Sur) in the intel x64 architecture

- 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:

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 (January 2021).
Chapter 5. 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.

5.1. 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

5.2. 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 Mac clients can be found in the directory `/etc/opsi/package-updater.repos.d/` in the files `uib-mac-testing.repo` and `uib-mac.repo`.

If you do not have these files installed yet, you can find them under:

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

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

```ini
; This repository provides testing of opsi products to manage MacOS
; clients with opsi.
[repository_uib_macos_testing]
description = opsi macOS packages in testing
active = true
baseUrl = http://download.uib.de
dirs = opsi4.1/testing/packages/macos/localboot/
autoInstall = false
autoUpdate = true
autoSetup = false
; Set Proxy handler like: http://10.10.10.1:8080
proxy =
```
Chapter 5. Installing the minimal MacOS opsi products.

/\etc/opsi/package-updater.repos.d/uib-mac.repo

; This repository provides table opsi products for managing MacOS
; clients with opsi.

[repository_uib_macos_stable]
description = opsi macOS packages in stable
active = true
baseUrl = http://download.uib.de
dirs = opsi4.1/stable/packages/macos/localboot/
autoInstall = false
autoUpdate = true
autoSetup = false
proxy = http://10.10.10.1:8080

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

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

or

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:

opsi-package-updater -v update

More information about opsi-package-updater can be found in the manual.

5.3. Manual installation of the MacOS opsi products.

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/opsi4.1/stable/packages/macos/localboot or at https://download.uib.de/opsi4.1/testing/packages/macos/localboot.

We recommend saving the .opsi files to /var/lib/opsi/repository. To ensure that opsinconfd 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.
Chapter 6. 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, 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 Mac 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.

6.1. Using service_setup.sh on macOS

6.1.1. 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.

```
sudo su
cd /Users/<username>
mkdir <mnt>
mount_smbfs //<serviceuser>@<opsi-server>/opsi_depot /Users/<username>/<mnt>
cd /Users/<username>/<mnt>/opsi-mac-client-agent
./service_setup.sh
cd
umount /Users/<username>/<mnt>
```

without password query

```
sudo su
cd /Users/<username>
mkdir <mnt>
mount_smbfs -N //<serviceuser>:<servicepass>@<opsi-server>/opsi_depot /Users/<username>/<mnt>
cd /Users/<username>/<mnt>/opsi-mac-client-agent
./service_setup.sh
cd
umount /Users/<username>/<mnt>
```

Example:
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-config-server), 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.

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

Example:

```
sudo su
mount_smbfs -N //adminuser@sepia/opsi_depot /Users/uib/opsi_depot
```

```
cd /Users/uib/opsi_depot/opsi-mac-client-agent
./service_setup.sh
```

```
cd
```

```
mount_smbfs -N adminuser:linux123@sepia/opsi_depot /Users/uib/opsi_depot
```

```
cd /Users/uib/opsi_depot/opsi-mac-client-agent
./service_setup.sh
```

```
cd
```

```
mount_smbfs -N //adminuser@sepia/opsi_depot /Users/uib/opsi_depot
```

```
cd /Users/uib/opsi_depot/opsi-mac-client-agent
./service_setup.sh
```

```
cd
```

```
mount_smbfs -N //adminuser@sepia/opsi_depot /Users/uib/opsi_depot
```

```
cd /Users/uib/opsi_depot/opsi-mac-client-agent
./service_setup.sh
```

```
cd
```

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-config-server), 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. The reboot is not triggered automatically.
6.2. 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':

![System preferences / sharing](image)

On the command line, this can be done as follows:

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

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

```
sudo systemsetup -getremotelogin
```

Disabling ssh access on the command line looks like this:

```
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:

```
```

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:

```
cd /var/lib/opsi/depot/opsi-mac-client-agent
```

Typical calls are:

- For a single client:
Chapter 6. Integration of existing MacOS clients into opsi.

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
Chapter 7. 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.

7.1. 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.

7.2. 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-configed', '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 please 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.

7.3. 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-configed', 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.

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-javavm product.
7.4. Distribution of opsi standard products: opsi-configed.

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-configed', 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**.
Chapter 8. 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.

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

8.1.1. Introduction

This tutorial cannot replace training or studying of the manuals. It only serves as an introduction. Therefore as first the reference to further sources:

trainings:
The uib GmbH offers opsi trainings in Mainz and in-house trainings:
https://uib.de/de/support-schulung/schulung/

manuals:
https://uib.de/de/opsi-dokumentation/dokumentationen/
Specially important:

wiki (scripts, tips, links):
https://forum.opsi.org/wiki

support forum:
check https://forum.opsi.org

8.1.2. 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 according 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-gui.app
|--Contents
 ||--Info.plist
 ||--PkgInfo
 ||--MacOS
 | ||--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.

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

First an example for a simple opsi-script script:

```ini
[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`.

### 8.1.4. 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 else endif statements**
- **for loops over string lists**
- **Functions**
Figure 1. Avoiding duplicate code via distributed sub

8.1.5. 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.

8.1.6. 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:
8.1.7. 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")
```

8.1.8. Elementary commands for primary sections

**String-Variable**

**Variable declaration**

`DefVar <variable name>`

**Variable assignment**

`Set <variable name> = <value>`

**Example:**

```
DefVar $ProductId$
Set $ProductId$ = "firefox"
```
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 [else] endif

Syntax:

if <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:

```bash
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]

See also:

In the following chapters special opsi MacOS commands to install software are presented, which come from the opsi-script library uib_macosinstalllib. This documentation is in English because it 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

\texttt{install\_macos\_app}($\texttt{myapp}$ : string) : string

Description

try to install the app given by \texttt{myapp}$

- Parameter: \texttt{myapp}$
  - Type: \texttt{String}  - Calltype: \texttt{CallByValue}
    - Parameter \texttt{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
- Copyright: AGPLv3

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 succseeded"
else
  LogError "installation failed"
endif
```

Documentation of local function install_macos_pkg

Definition

\texttt{install\_macos\_pkg}($\texttt{mypkg}$ : string) : string

Description

try to install the pkg file given by \texttt{mypkg}$

- Parameter: \texttt{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:

```python
[Actions]
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 succseeded"
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 succseeded"
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 succseeded"
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:

```plaintext
see: install_macos_generic
[Actions]
importlib "uib_macosinstalllib"
DefVar $installfile$
DefVar $installresult$
comment "we have a *.* and install with install_macos_generic"
set $installfilename$ = "%scriptpath%/files/opsi-script.app"
set $installresult$ = install_macos_generic($installfilename$)
if stringtobool($installresult$)
  comment "installation succeeded"
else
  Error "installation failed"
endif
```

8.1.9. Third example: MacOS template 'm-opsi-template'.

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

```
setup.opsiscript: Installationsscript

; import complete file !
importlib "uib_exitcode.opsiscript"
importlib "%scriptpath%/osd-lib.opsiscript"
importlib "%scriptpath%/uib_macosinstalllib.opsiscript"
```
Chapter 8. Integration of own software into the software distribution of opsi.

; $ProductId$ 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/<path to the product>"
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>"
;-------------------------------------------------------------
Sub_check_installSuccess
comment "Copy files"
Files_install
[Sub_check_installSuccess]
if $installSuccess$ = "false"
  LogError "Installation failed"
  isFatalError "Installation failed"
else
  Comment "Installation success"
endif
[Files_install]
; Example of recursively copying some files into the installation directory:
; copy -a "%ScriptPath\files\".*" "$InstallDir$"
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\"

;-----------------------------------------------------
[Linkfolder_remove_desktop_icon]
; check delete_element
set basefolder common_desktopdirectory
set subfolder ""
delete_element $productId$

;-----------------------------------------------------
8.2. Create an opsi product package
8.2.1. Installation of the opsi-setup-detector, opsi PackageBuilder and opsi-logviewer

Installation des opsi PackageBuilder

The opsi PackageBuilder is currently available for Windows and Linux and MacOS.

The installation files / packages of the `opsi-package-builder` can be found here:

Dort finden sich im oberen Teil die Links auf die Installationspakete für Windows, Linux und MacOS.

The `opsi PackageBuilder` is not made by 'uib' but by Holger Pandel from the opsi-community (thank you!).

Der `opsi PackageBuilder` unterliegt einer OpenSource Lizenz:
https://github.com/pandel/opsiPackageBuilder/blob/master/LICENSE_DE

Der `opsi PackageBuilder` hat eine eigene Dokumentation welche mit installiert wird.

Installation of the opsi-setup-detector

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

You can install the opsi-setup-detector via opsi:

The package `opsi-setup-detector` belongs to the opsi standard products and should be installed on your opsi-server. If not, use:

```
opsi-package-updater install opsi-setup-detector
```

to install it on the opsi-server.

A setup program to install the opsi-setup-detector without opsi can be found at:
https://download.uib.de/opsi4.1/misc/helper/

The functionality of the Linux version is limited, since certain programs for analyzing setup programs from certain frameworks are not available for Linux:

- Detailed analysis of Inno-Setups due to the lack of innounpack.exe for Linux
- Detailed analysis of wix-setups due to the lack of dark.exe for Linux

The opsi product `opsi-setup-detector` has a dependency on the opsi product `opsi-package-builder`. The `opsi-setup-detector` uses the `opsi-package-builder` if available, but can for the most part also work without it. The installation of the `opsi-package-builder` is recommended.

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/opsi4.1/misc/helper/

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

---

Chapter 8. Integration of own software into the software distribution of opsi.
8.2.2. 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.

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 2. opsi-setup-detector Necessary configuration on first startup](image-url)
Figure 3. 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:

- Windows
- Linux
- MacOS
- multi platform

The offered tasks for MacOS:

1. **Analyze single MacOS Installer file and generate an opsi package**
   Here it starts from a MacOS installer file and then goes through the whole process to create an opsi package. This process is described in the next chapter.

2. **Create an opsi package template for MacOS**
   This point does not ask for a setup file, but creates a template comparable to the opsi product `opsi-template` with the exception that here the specifications from the product configuration are already taken over.

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 single setup file and create an opsi package**

The workflow is here described using **Analyze single setup file and create an opsi package** as an example.
After you selected the task, you will get a file selection dialog. Select now the setup file that has to be analyzed. The analyze will start directly after the selection is done.

**Opsi-setup-detector: Analyze**

If the analyze found no result, you will get here a **Sorry unknown Installer**. If we have a successful analyze, you will see the result page.
• Detected Setup Type: Type of detected Installer
• MST allowed:
• Link with information about the installer
• Setup file: Path and name of the analyzed setup file
• MST file: For MSI installers or installers which contain MSI, an MST file can be specified here which will be integrated into the MSI call.
• Msid: For MSI installers or installers that contain MSI in the form of product code
• Software version: The version of the software to be installed if determinable.
• Setup file size MB: Size of the setup file in MB
• Required space MB: This value is an estimate of six times the size of the setup file and can be adjusted if necessary
• InstallDir: As far as detected the directory where the software will be installed.
• Unattended installation command: The determined command for a non-interactive installation.
• Unattended deinstallation command: The determined command for a non-interactive deinstallation.
• Deinstallations program: The determined deinstallations program

The values determined here can now be corrected or supplemented if necessary. The button Next Step leads to the first page of the product configuration. The metadata of the opsi product to be created is entered here.

The values determined here can be incorrect and are probably incomplete!
After an initial installation, you should definitely check the values of InstallDir, Check deinstallation program and software version and adjust them in your script if necessary.

Figure 6. opsi-setup-detector Result of the analysis

Opsi-setup-detector: Product configuration 1
Chapter 8. Integration of own software into the software distribution of opsi.

Figure 7. 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**

Figure 8. opsi-setup-detector Product configuration 2
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.

**Productid**
- Productid (identifier) of the product to which a dependency exists.

**Dependency Mode**
- You can either request the **Action** setup or (see below) the **Status** (installed).

**Action or Status**
- For **Status**: Status that the product to which a dependency corresponds, should have (installed). If there is another status, the product is set to setup.
- For **Action**: Action request, which should be set on the product, whereupon there is a dependency (setup).

**Dependency type**
- Installation order. If the product for which there is a dependency must be installed before the installation of the current product, then this is before. If it must be installed after the current product, this is after. If the order doesn’t matter then nothing has to be entered here.

**Hint:**

Unfortunately there is currently no generic mechanism for uninstalling product dependencies. The Product Dependency mechanism is only reliable for action: setup and the (before- or after-) setup actions to be triggered and installed status, in the case of a 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 10. opsi-setup-detector Property Editor](image)

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

To select another icon, use the button Open icon directory to select the directory in which you expect to find icons. As a preselection you get a directory of 'open source' icons: 128x128, supplied with the opsi-setup-detector. Select a subdirectory and the icons will be displayed.

Now you can select an icon from the display.

After the product configuration is performed, the product can be created.

**Opsi-setup-detector: Create product**
• 'Path to opsi-workbench' is a drive letter or UNC path where the share ops\workbench of your opsi-server is mounted.

• For security reasons, an opsi package can only be rebuilt if it doesn’t exist already. If an existing package is to be overwritten, the directory must first be deleted from the opsi Workbench.

To the left of the button 'Create opsi package' there are three possible options, which refer to the function of the button:

Create Mode’ is a selection area where you can specify what happens when creating the package:

* Create opsi product files creates the directory tree for the new opsi package on the selected ops\workbench if it does not already exist. The files required for the package will be created or copied.

* Create opsi product files and build package performs the operations described in the first point. Additionally, the opsi Package Builder is called to create the opsi package from the created directory tree. The exact processes are determined by the selection field Tree mode:
  - Only build starts the opsi Package Builder without interactive GUI, creates an opsi package from the directory tree via server command opsi-makepackage and terminates the opsi Package Builder after work is done.
  - build and install starts the opsi Package Builder without interactive GUI, creates from the directory tree via server command opsi-makepackage an opsi package installs the package via server command opsi-package-manager and finishes the opsi Package Builder after the work is done.

* Create opsi product files and start interactive package builder performs the operations listed in the first item. Additionally the opsi Package Builder is called interactively. You have to quit it yourself to return to the opsi-setup-detector. For installation, configuration and operation of the community project 'opsi Package Builder' check https://forum.opsi.org/viewforum.php?f=22

* Create opsi package is the button that initiates the package creation. For security reasons, an opsi package can only be rebuilt if it doesn’t exist already. If a package with this name already exists, you will be asked if the existing directory may be deleted.


8.2.3. 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](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:
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: /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'.
In this tab you see on the left side the general metadata of the opsi product as you have already been explained in Section 8.2.2.4, “Opsi-setup-detector: Product configuration 1”.

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

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).
8.2.4. 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 `winst32.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-winst\winst32.exe`. If not, copy the `opsi-winst` directory from the share `\<opsiserver\opsi_depot`, from the `opsi-winst\files` directory.

- **Linux:** start file `/usr/bin/opsi-script-gui`.

- **MacOS:** Start the application `/Applications/opsi-script-gui`.

You'll see the following window:

![opsi-script-gui](image)

Figure 26. 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 to 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.

### 8.2.5. 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-depot-server, ‘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:
8.2.6. 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>
```


8.2.7. Example of a 'control' file

```
[Package]
version: 1
depends:

[Product]
type: localboot
id: mytest
name: My Test
description: A test product
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
```
Chapter 9. General notes about MacOS

9.1. The Keyboard under MacOS

Table 1. Apple Tasten

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

Table 2. 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>&quot;</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>&gt;</td>
<td>Single right angle quotation mark</td>
</tr>
</tbody>
</table>
9.2. 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:

```
[ShellInAnIcon_brew_install_java]  
; https://github.com/SAP/SapMachine/wiki/Installation
EXITCODE=0
sudo -H -u opsisetupadmin /usr/local/bin/brew tap sap/sapmachine
EC=$?; if [ $EC -ne 0 ]; then EXITCODE=$EC; fi
sudo -H -u opsisetupadmin /usr/local/bin/brew install --cask sapmachine-jdk
EC=$?; if [ $EC -ne 0 ]; then EXITCODE=$EC; fi
exit $EXITCODE
```
To find out if the installation of a desired product is supported via homebrew, it is best to use the following website:
https://formulae.brew.sh/

Helpful links for homebrew:
https://formulae.brew.sh/

Additional helpful Links for homebrew:
https://docs.brew.sh/
https://devhints.io/homebrew

9.3. 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.

9.4. 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
• /var/opsisetupadmin (Home directory opsisetupadmin)
• /var/lib/opsi-client-agent
• /Library/LaunchDaemons/ (Start configuration for opsi daemons)
• /Library/LaunchAgents/ (Start configuration for opsi-systray)

9.5. 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`
Chapter 10. 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 die opsi Community.

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