

# **OpenSLX - Step-By-Step-Howto - OpenSLX Lab**

# Table of Contents

<b>OpenSLX</b> .....	<b>1</b>
Wiki.....	1
<b>Step By Step HOWTO: Stateless Linux with OpenSLX</b> .....	<b>2</b>
Installation.....	2
Functionality.....	3
SLX-Server.....	4
Program names und global Settings.....	5
Stage1 from Network Sources or reference machines.....	7
Plug-ins - Modular Extension of the Base Package.....	8
System Export.....	8
Quick Start.....	9
Database and the Boot Setup.....	10
Communication with the Database.....	11
System-Update and Extension.....	12
Cleanup.....	13
Boot Cycle.....	13
PXE-Boot.....	13
DHCP for the IP-Address assignment.....	14
TFTP.....	14
Syslinux.....	15
Configuration of OpenSLX Clients.....	16
The Kernel Command line.....	16
Central Configuration File machine-setup.....	17
Problems.....	17
Further Development.....	18

# OpenSLX

- [Overview](#)
- [Activity](#)
- [Roadmap](#)
- [Issues](#)
- [Wiki](#)
- [Repository](#)

## Wiki

[Start page](#)

[Index by title](#)

[Index by date](#)

[History](#)

# Step By Step HOWTO: Stateless Linux with OpenSLX

The operation and administration of a larger number of computers is quite a lot of effort. This is also true for Linux installations. So that administration won't become the main task with your implementation and maintenance of a larger installation, there is a need for special technologies to get the problem under control. Apart from the classic method of Software Deployments and Patch Managements, Stateless approaches have the best chance to relieve the admin from boring routine tasks. On request, one installation can serve any number of Client-machines over the network. For this there is no need for any special preparation of the machines: PXE or Etherboot are sufficient to allow the machine to boot from the network. From this, additional freedom of design is achieved: Local installations don't need to loose ground, but can for example, during migration or mixed scenarios, offer an alternative to the Stateless operation.

OpenSLX deals with GPL-Software that allows installation and easy administration on hard drive free Linux Workstations. The project focuses of large installations with a similar deployment profile (So not necessarily machines with special hardware for production control etc.). This is currently implemented in Schools or Universities to have a highly flexible and stable pool environment for training courses, classes etc. On the established Stateless-Linux-Basis, another OS for training purposes, such as Windows or Linux, can effectively be distributed through virtualization technologies (currently VMplayer, later also VirtualBox or XEN).

The switch-over between different operation modes occurs at boot time, since no costly Image distribution (Imagecopy) occurs before or during the machine boot.

Plenty of good Linux distributions are available: For this reason OpenSLX doesn't choose the approach to offer its own package collection. Instead a script collection is available which many standard distributions prepare for the stateless operation. The OpenSLX package prepares a middleware for you that stays hidden from the user of a machine. So the appropriate "Look & Feel" of a distribution is maintained, the customization and changes occur in the background.

Users use their desktop to access all of their machines available equipment as usual: There are no special requirements since everything occurs locally and no costly, for example, Auto-mount chains need to happen. Playback of movies or use of 3D-applications is also no problem. Furthermore, only this way, can the extensive performance potential of today's desktop PCs be fully used and be given exclusively to the user.

This type of Linux Desktop allows the familiar uses of versatile applications: Virtualization environments such as Virtual Box, QEMU/KVM or VMware(Player) allow you to boot further Operating Systems.

## Installation

The OpenSLX Package is actually like every other OpenSource Package, not finished, however is constantly being developed further since 1997. The current version is some pre 5.0. You can download it from the project homepage, [packages.openslx.org](http://packages.openslx.org), as RPM or Debian package.

For OpenSuSE choose TGZ, for Ubuntu and alike the DPKGs:

```
x60s:~ # wget  
x60s:~ # rpm -i
```

For current bug fixes, new features and further supporting distributions, you can anonymously access the sources in the Subversion:

```
x60s:~ # svn co http://svn.OpenSLX.org/svn/openslx/openslx
```

## OpenSLX - Step-By-Step-Howto - OpenSLX Lab

```
Ausgecheckt, Revision 3NNN.  
x60s:~ # cd openslx/trunk  
x60s:~/openslx/trunk # make install
```

In many cases you need to add some additional Perl packages: For e.g. Ubuntu you should look for `libclone-perl`, `libconfig-perl`, `libdbi-perl`, `libdbd-mysql-perl` and `libdbd-sqlite3-perl`. For RedHat or derivatives like Scientific Linux you might need to add the following RPMs: `perl-Clone`, `perl-Config-General`, `perl-DBD-SQLite`, (`perl-DBD-mysql`).

Irrespective of how you installed OpenSLX, OpenSLX will work with its original settings in the following directory structure:

```
/etc/opt/openslx  
/etc/opt/openslx/distro-info  
/opt/openslx  
/opt/openslx/lib  
/opt/openslx/share  
/opt/openslx/bin  
/srv/openslx  
/srv/openslx/export  
/srv/openslx/tftpboot  
/var/opt/openslx  
/var/opt/openslx/config  
/var/opt/openslx/db  
/var/opt/openslx/stage1
```

Only directories with the statistical SLX files are created, directories such as `/var/opt/openslx` or `/srv/openslx` are created at the setup of the Client environment. The command `slxsetting` allows you to specify:

```
x60s:~ # slxsettings set private-path='/var/opt/openslx' set public-path='/'  
setting private-path to '/var/opt/openslx'  
setting public-path to '/'
```

All global settings of the packages are placed beneath `/etc/opt/openslx`. All statistical files are placed in `/opt/openslx`, like scripts and the required Perl modules and Includes. All data that is generated during runtime is placed in `/var/opt/openslx`, if not changed through `slxsettings`. The directories are empty after installation. In config you can place files and configurations of later exported systems, db records the database of your Systems and Clients.

In a further step Exported systems are finally placed in `stage1`. While the amount of data in other directories can be disregarded, you should expect at least 5 - 50 GBytes of data depending on the type and number of OpenSLX-Installations (`stage1`).

If the specification of the directories isn't what you imagined, you can adjust these using the script `slxsettings`. This way you can for example specify that the `Stage1` files aren't stored on your Server-Rootfilesystem but on a separately mounted partition instead.

See the current version of your OpenSLX-Toolsets through:

```
x60s:~ # slxversion  
5___.0.0revNNNN
```

## Functionality

The task of the OpenSLX project is the setup of exportable systems; Distributions in a certain version, that can incorporate stateless clients over the network as Rootfilesystem. An SLX server can supply many of these systems.

For a good overview, the setup procedure of your stateless is separated into many clear stages:

- Stage0 identifies a running reference system on real or virtual hardware, of which a copy through Rsync is generated on the server. This stage simply plays a role for the clone mode.
- Stage1 identifies a executable client system, which is placed beneath `/var/opt/openslx/stage1/<name>`. With consistent hardware architecture you can access this directory through "slxos-setup shell <name>". The system is strictly setup and extended with particular local extensions. To this you could count the installation of special, not offered from the distribution, external software packages. There are two ways in which you could setup Stage1, through direct installation from the respective packet source or copy of an existing system (Clone, Stage0).
- Stage2 - from stage1, different Client-Rootfilesystems can be created. For this stage1 is duplicated, this means the omission of irrelevant files for the stateless operation, a new sub-directory in NFS-Export is created or for Network block devices a compressed SquashFS is applied. An appropriate Kernel is needed for this (usually directly from the distribution being used), at least one InitialRamFS fitting to the Kernel and one Client configuration. The Client configuration can contain an array of files that are extracted to the appropriate place in Stage3.
- Stage3 - During Stage0-2 administrative procedures occur on the SLX server, which you only need to do a few times. Stage3 occurs on every single client at every restart within the InitialRamFS. Here the individual machines are setup. This area is covered by the shell scripts `init` (SLX-Init), `hwautocfg` and `servconfig`. The latter places all the clients important configuration files in `/mnt/etc`. Local extensions (for all clients that use a certain InitRamFS) are possible by using `preinit.local` at the very beginning of SLX-Init, `postinit.local` towards the end of SLX-Init. `postinit.local` is created through `ConfTGZ`. This is a package of configuration files that are downloaded through, for example, TFTP and then extracted in Stage3. The template for this is in `/var/opt/openslx/config/<system>`. `<system>` is the name of the Vender OS linked through two colons and the type of the export.\* In Stage4, SLX view, only some smaller configuration tasks occur. The client now has a complete Rootfilesystem, all programs and files, so now the keyboard configuration can occur. Also, in this stage, the graphical login is started to an appropriate earlier time.

Depending on the type of the clients, you may want to prepare one or more different system exports on a server. This can mean that a Stage1 system in many varieties, like NFS and at the same time NBD/SquashFS, should be delivered to the clients. On the other hand, very different Stage1's can be served with the same technology from different exports. This is, for example, useful for on-site tests, when you would like to offer the new system parallel with an existing system.

There are two options if you would like to export one and the same base system in different ranges: You either, from the start, put two different Stage1 versions into place which are maintained separately. Or you move the differences into a certain area which will later be additionally mounted in the environment, for example by using a Stacking file system.

## SLX-Server

All the considerations made so far have an effect on space requirements. Depending on the range of the installed system, a Stage1 version needs 2 - 8 GBytes of disk space in the folder `/var/opt/openslx/stage1`. A Stage2-NFS-Export requires between 90 and 98% of this amount, a NBD/SquashFS-Export between 35 - 50%. Accordingly you should ensure `/srv/openslx/export` has enough disk space.

SLX-Clients operate stateless and have no local operating system installed. They obtain their joint Rootfilesystem from a central server, which should be configured with sufficient capacity. The largest load is carried by the Rootfilesystem export service, which generates the largest Filesystem output. For a setup consisting of 100 clients, you should have one to two Gigabit Interfaces and fast hard drives bound by RAID installed on your server. Since only the files in the area of the exports of the clients are incorporated, you can place `/var/opt/openslx/stage1` on slower drives. The requirement of DHCP and TFTP do not even burden an average server with more than 1000 clients. While the server sits better on the backbone of your network, a

100Mbits interface is sufficient for the clients. Depending on the type of operation, in one day between 0.3 and 3 Gigabytes of data is transported per client. The installation of Network block devices with SquashFS can more than halve the amount of data compared to NFS.

An increase of the client connections to 1 Gigabit should be accompanied with improved IO of the server: A very fast RAID or placing the exports in RAM connected with Gigabit-Trunking accelerates the start and operation of the clients considerably. They are then generally noticeably faster as in classic operation from a local hard disk.

For smaller network of demonstration with few clients, an average equipped fileserver of normal Desktop PC should suffice.

## Program names und global Settings

The Configuration and Setup programs all have the prefix "slx", most of which are Perl scripts. Most tools deliver some short help with the Option "--help", the Manpage can be accessed through "--man".

You can set global settings after the installation using slxsettings, like directory structure. Calling slxsettings without parameters, a list of current settings is shown:

```
x60s:~ # slxsettings
paths fixed at installation time:

    base-path='/opt/openslx'
    config-path='/etc/opt/openslx'
current base settings (cmdline options):
    db-name='openslx'
    db-spec=''
    db-type='SQLite'
    debug-confess=''
    locale='de_DE.UTF8'
    locale-charmap='UTF-8'
    private-path='/var/opt/openslx'
    public-path='/'
    temp-path='/tmp'
    verbose-level=''
extended settings:
    db-passwd=<unset>
    db-user=<unset>
    default-shell='bash'
    default-timezone='Europe/Berlin'

    mirrors-preferred-top-level-domain='de'
    mirrors-to-try-count='20'
    mirrors-to-use-count='5'
    ossetup-max-try-count='5'
    pxe-theme=<unset>
    pxe-theme-menu-margin='9'
```

slxsettings provides through an extensive help (slxsettings --man), which summaries all Perl based SLX commands:

```
SLXSETTINGS(1)          User Contributed Perl Documentation          SLXSETTINGS(1)

NAME
    slxsettings - OpenSLX-script to show & change local settings

SYNOPSIS
    slxsettings [options] [action ...]

Script Actions
```

## OpenSLX - Step-By-Step-Howto - OpenSLX Lab

```
set <option-name=value>      sets the option to the given value
reset <option-name>          resets the given option to its default
```

### List of Known Option Names

```
db-name=<string>              name of database
db-spec=<string>              full DBI-specification of database
db-type=<string>              type of database to connect to
locale=<string>               locale to use for translations
logfile=<string>              file to write logging output to
private-path=<string>         path to private data
public-path=<string>          path to public (client-accessible) data
temp-path=<string>            path to temporary data
verbose-level=<int>           level of logging verbosity (0-3)
```

### General Options

```
--help                        brief help message
--man                          full documentation
--quiet                        do not print anything
--version                       show version
```

### Actions

```
set <openslx-option>=<value>
    sets the specified option to the given value

reset <setting>
    removes the given setting from the local settings (resets it to
    its default value)
...

```

Thus the Debug expenditure can be controlled by the option `--verbose-level=0-3` with all SLX commands. Changes of the values can be made with `set`, `reset` allows for resetting on the default value, if available, or deletes the entry.

For most SLX environments the default values should be ok. If however you have for example mounted a large empty partition under `/data/openslx` and would like to use this for OpenSLX, you can adjust the Stage1-Installationen on these:

```
x60s:~ # slxsettings set public-path='/data/openslx'
setting public-path to '/data/openslx'
x60s:~ # slxsettings set mirrors-preferred-top-level-domain=de
setting mirrors-preferred-top-level-domain to 'de'
```

The second example shows an important setting for the search of mirrors of the sources of installation of the different distributions. OpenSLX tries to determine the Domain with the installation, however fails in networks with private Domain names. Likewise you can change the type of database which administers with OpenSLX exported systems and your clients. The standard type is "SQLite", depending upon installed Perl components you may alternatively also use MySQL.

```
x60s:~ # slxsettings set db-type=SQLite
setting db-type to 'SQLite'
```

Once you have decided on the type of database, you cannot easily change it in the running system. After base installation, you must devote to a database at Stage0 or Stage1.

You will find the settings in `/etc/opt/openslx/settings`:

```
SLX_DB_TYPE=SQLite
SLX_MIRRORS_PREFERRED_TOP_LEVEL_DOMAIN=de
```

```
SLX_PUBLIC_PATH=/data/openslx
```

## Stage1 from Network Sources or reference machines

Depending on the Linux Distribution, there are one or two ways to setup Stage1: either you install a fresh system from the package sources or you clone an already setup machine, which is running under your Linux distribution. Both are taken care of by the command `slxos-setup`, which is made up of "`slxos-setup command <parameter>`". Information is listed by the subcommand "`list-supported`":

```
x60s:~ # slxos-setup list-supported
gentoo-2007.X          (clone)
suse-10.2              (clone,install)
suse-10.2_x86_64      (clone,install)
suse-10.3             (clone)
suse-10.3_x86_64      (clone)
suse-11.0             (clone,install)
suse-11.0_x86_64      (clone,install)
ubuntu-8.04           (clone,install)
ubuntu-8.04_amd64     (clone,install)
```

So far not all listed distributions from SLX-Stage3 are supported. New installations are performed by `slxos-setup` with the subcommand "`install`", followed by the distribution:

```
x60s:~ # slxos-setup install suse-11.0
[ ... lots of yum output ... ]
Total download size: 69 k
Downloading Packages:
(1/2): nbd-2.8.7-14.i586. 100% |=====| 45 kB    00:00
(2/2): squashfs-kmp-defau 100% |=====| 24 kB    00:00
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing: squashfs-kmp-default      ##### [1/2]
  Installing: nbd                      ##### [2/2]

Installed: nbd.i586 0:2.8.7-14 squashfs-kmp-default.i586 0:3.1_2.6.18.2_34-34
Complete!
Starting SuSEconfig, the SuSE Configuration Tool...
Running in full featured mode.
Reading /etc/sysconfig and updating the system...
Executing /sbin/conf.d/SuSEconfig.perl...
Executing /sbin/conf.d/SuSEconfig.permissions...
Finished.
Vendor-OS 'suse-11.0' installed succesfully.
Vendor-OS 'suse-11.0' has been added to DB (ID=1).
```

Depending on bandwidth and load of your network you can plan a coffee or lunch break till this step is finished. All of the installed distributions can be refreshed with "`slxos-setup update <distro>`", which is considerably faster than the primary installation.

If you already have an established system, since, for example, you may want to turn Fat-Clients into OpenSLX-Clients, Clone setup may be a good option. The `slxos-setup` subcommand "`clone`" requires the outgoing source. To be able to identify a classic cloned system of a classic RPM installation, give the child an individual name. Everything you enter after the distribution name and version, for example "`-mysystem`", become part of the name. The actual clone operation occurs in the background with the command `Rsync`, so that at the start you are prompted for the root password of the reference machine. On an Ubuntu Reference system you must ensure that "`root`" can login through SSH.

## OpenSLX - Step-By-Step-Howto - OpenSLX Lab

```
x60s:~ # slxos-setup clone 10.60.4.50:/ suse-11.0-mysystem
Cloning vendor-OS from '10.60.4.50:/' to '/space/stage1/suse-11.0-mysystem'...
Password:
receiving file list ...
...
var/yp/
var/yp/nicknames
var/yp/binding/

sent 5765235 bytes received 6044617832 bytes 5710602.23 bytes/sec
total size is 6025777602 speedup is 1.00
Vendor-OS 'suse-11.0-mysystem' has been cloned successfully.
Vendor-OS 'suse-11.0-mysystem' has been added to DB (ID=3).
```

The subcommand "list-installed" shows you all existing Stage1-Systems:

```
x60s:~ # slxos-setup list-installed
List of installed vendor-OSes:
    suse-11.0-mysystem
    suse-10.3
```

## Plug-ins - Modular Extension of the Base Package

The general Software Architecture of OpenSLX aims to extend the base functionality of the system by implementing special extensions (OS-Plugins). The previous steps were sufficient for a bootable base system, however not a comfortable graphical desktop. This task is taken over by the plug-ins "xserver" and "desktop". The first takes care of the automatic setup of Xorg and the integration of proprietary OpenGL drivers from NVidia or ATI/AMD. The second deals with the setup of display managers and the graphical default session.

Installation of the plug-ins and their configuration, if you wish to change their default settings, is taken care by the command `slxos-plugin`:

```
x60s:~ # slxos-plugin install suse-11.0 desktop manager=kdm
Plugin desktop has been installed into vendor-OS 'suse-11.0'.
x60s:~ # slxos-plugin install ubuntu-8.04-clone xserver
Plugin xserver has been installed into vendor-OS 'suse-11.0'.
```

For a later post installation of plug-ins: Without a following export and call to the OpenSLX Demuxers, which are explained later, the selected plug-in won't be available on the booting SLX-Clients.

Stage3-Attribute, that like the desktop plug-in includes settings for later running clients, writes `slxos-plugin` into the database. There you can customize these anytime for single clients, whole groups or vendor OS.

## System Export

The previous operations have not yet covered the allocation of a Rootfilesystem to booting clients. This occurs in the next step with `slxos-export`, which realises the transition from Stage1-Installations into Stage2-Exports. The parameter "list-installed" should, when called with `slxos-export`, give the identical result as with `slxos-setup`:

```
x60s:~ # slxos-export list-installed
List of installed vendor-OSes:
    suse-10.3-mysystem
    suse-11.0
```

From this system you can now create an Export, currently supported are "nfs" and "nbd-squash". Further types will be added in future development. The subcommand "export", you create a new export, for example:

## OpenSLX - Step-By-Step-Howto - OpenSLX Lab

```
x60s:~ # slxos-export export suse-11.0-x60 sqfs-nbd
invoking mksquashfs...
Parallel mksquashfs: Using 1 processor
Creating little endian 3.0 filesystem on /data/openslx/export/sqfs/suse-11.0-x60
  block size 65536.
[=====] 294751/294751 100%
[ ... ]
Filesystem size 2325395.23 Kbytes (2270.89 Mbytes)
  41.13% of uncompressed filesystem size (5654449.42 Kbytes)
Inode table size 2888177 bytes (2820.49 Kbytes)
  30.34% of uncompressed inode table size (9518178 bytes)
Directory table size 2735640 bytes (2671.52 Kbytes)
  47.98% of uncompressed directory table size (5701677 bytes)
[ ... ]
vendor-OS '/data/openslx/stage1/suse-11.0-x60' successfully exported to '/data/openslx/export/sqf
Export 'suse-11.0-x60:sqfs-nbd' has been added to DB (ID=1)...
#####
Please make sure you start a corresponding nbd-server:
  nbd-server 5000 /data/openslx/export/sqfs/suse-11.0-x60 -r
#####
system 'suse-11.0-x60:sqfs-nbd' has been successfully added to DB (ID=1)
```

A classic NFS Rootfilesystem can be created through a similar call:

```
x60s:~ # slxos-export export suse-11.0 nfs
building file list ...
...
... rsync output ...
...
sent 5642940230 bytes received 5681918 bytes 9147566.23 bytes/sec
total size is 5624419842 speedup is 1.00
vendor-OS '/data/openslx/stage1/suse-11.0-x60' successfully exported to
'/data/export/nfs/suse-11.0'!
Export 'suse-11.0-nfs' has been added to DB (ID=1)...
#####
Please make sure the following line is contained in /etc/exports
in order to activate the NFS-export of this vendor-OS:
  /data/export/nfs/suse-11.0 *(ro,no_root_squash,async,no_subtree_check)
#####
system 'suse-11.0-nfs' has been successfully added to DB (ID=1)
```

Afterwards you can ensure that everything has worked through calling "slxos-export list-exported".

## Quick Start

Up to this point you prepared the Rootfilesystem (either classic NFS or NBD with SquashFS) and the database filled with some base data. Missing now is the preparation of the respective Kernels with the appropriate InitialRamFS and one matching client configuration. This task is taken care by the command slxconfig-demuxer, which doesn't require any further parameters.

It works directly with the entries in the database. With the base data you already have a valid configuration for each requesting client. For more specific settings, the tool slxconfig can be used, which will be explained in the next section.

Your ISC-DHCP-Server should be running and the following two entries for the domain from which you want to boot stateless:

```
next-server 10.8.4.254;
filename "pxe/pxelinux.0";
```

The second entry depends on the specified TFTP root directory. How to proceed: When DHCP is running, enter the appropriate values. Now you should have prepared everything needed to test the first clients. If you have not yet turned on the PXE-Bootability, do this on your clients now. They should now be assigned an IP Address at start-up and then download the Kernel and InitRamFS.

## Database and the Boot Setup

As soon as you want to manage multiple systems or give clients varying settings, the configuration through the database comes into play. To communicate with the database, which is located by `slxox-*` under `/var/opt/openslx/db` or where ever you specified, use the command `slxconfig`. It can occur that not every type of database is supported on every server. This will be determined at the end of the call `slxos-setup`. Then you can edit this using `slxsettings`.

This plans to be extended to a web interface in future versions. Entries in the database up until now were entered automatically and show:

```
x60s:~ # slxconfig list-vendoros
List of the matching vendor-OSes:
    suse-10.3-mysystem
    suse-11.0
x60s:~ # slxconfig list-export
List of the matching exports:
    suse-10.3-mysystem      (nfs)
    suse-11.0:sqfs-nbd     (sqfs-nbd)
```

For the installation Vendor-OS, called SLX-Stage1 and the exports created from this. Generally one entry will be created for one system in the database:

```
x60s:~ # slxconfig list-systems
List of systems:
    <<<default>>>
    suse-10.3-mysystem:nfs
    suse-11.0:sqfs-nbd
```

now displays

```
x60s:~ # slxconfig add-client slx-test01 mac=00:01:02:03:04:06
client 'slx-test01' has been successfully added to DB (ID=1)
x60s:~ # slxconfig add-client slx-test02 mac=00:11:22:33:44:55
client 'slx-test02' has been successfully added to DB (ID=2)
```

So that your Administrator can also login on the clients, you should determine a joint root password. For German users it makes sense to change the default language:

```
x60s:~ # slxconfig change-system suse-11.0:sqfs-nbd country=de
```

If everything works then you can remove this again. One alternative would be to distribute an "Authorized"-SSH-Key to the clients using `ConfTGZ`, to be able to access a fixed machine without problems. Debugging is often not possible without access to the client itself.

You complete the last step by calling `slxconfig-demuxer`. This command does not require any further subcommands or parameters, since it obtains all of its information from the database. All the necessary operations to complete the Stage2 phase are done in this step.

The `slxconfig-demuxer` completely rebuilds the directory `<public-path>/tftpboot` and `<public-path>/tftpboot/client-config`. And changes that you may have made in these folders will be lost. There are multiple sources of the demuxers:

- Kernel, -modules come from Stage1, the respective Vender-OS from which an export was created
- The base configuration, the InitRamFS setup, for the created InitRamFS will be generated from the database. Entries can be edited with slxconfig.
- The package of the configuration data (ConfTGZ) comes from /var/opt/openslx/config/<system-name>. The system name is automatically entered into the database by slxos-export.

## Communication with the Database

The tool slxconfig provides the user interface to the database. With this you can display all settings and make changes. "slxconfig list-system" shows you all activated, ready for potential clients, SLX systems.

In the last section you entered single clients into the database. You can see all set variables with:

```
x60s:~ # slxconfig --verbose list-system name="<system>"
List of systems:
system 'suse-11.0-main:nfs':
  clients      = <<<default>>>
  comment      = -
  description   = -
  export_id    = 6 (suse-11.0-main:nfs)
  hidden       = -
  id           = 5
  kernel       = vmlinuz
  kernel_params = aufs vga=0x317 quiet
  label        = suse-11.0-main:nfs
PLUGINS:
  30...Theme
  50...VMware
ATTRIBUTES:
  country      = de
  ramfs_fsmods = aufs
  ramfs_nicmods = forcedeth e1000 e100 tg3 r8169 pcnet32 b44 8139too
  slxgrp       = default
  start_cron   = yes
  start_dresnal = yes
  start_nfsv4  = yes
  start_ntp    = yes
  start_sshd   = yes
  start_x      = yes
  start_xdmcp  = kdm
  theme::name  = openslx
  vmware      = nfs://10.4.6.1/vol/vmwareImages/vmware
```

Most of the settings aren't assigned a value for the setup of the machine so they are assigned "<>". You can change the parameter of the system and default system:

```
x60s:~ # slxconfig change-system "<<<default>>>" start_xdmcp=gdm
system '<<<default>>>' has been successfully changed.
```

This configuration ensures that all cloned SuSE-11.0 clients start with the Default-Splash with the resolution 1024x768:

```
x60s:~ # slxconfig change-system suse-10.3-mysystem:sqfs-nbd \
kernel_params="vga=0x317 quiet"
```

The following database entry is needed if the home directory of the user is not prepared the secure NFSv3 way:

% predicted to be in plug-in

## OpenSLX - Step-By-Step-Howto - OpenSLX Lab

```
x60s:~ # slxconfig change-system suse-10.3-mysystem:sqfs-nbd automnt_dir=/home
automnt_src="nfs://11.00.4.2/home"
```

In many scenarios you may use different servers for DHCP, TFTP and the fileserver for the Client-Rootfilesystem. Should your export server differ from the DHCP-Server, you must inform your database:

```
x60s:~ # slxconfig change-export suse-10.3-main::sqfs-nbd server_ip=10.30.4.4 port=5002
export 'suse-10.3-main::sqfs-nbd' has been successfully changed
x60s:~ # slxconfig list-export suse-10.3-main::sqfs-nbd
List of exports:
export 'suse-10.3-main::sqfs-nbd':
  comment      = -
  id            = 10
  port         = 5002
  server_ip    = 10.30.4.4
  type         = sqfs-nbd
  uri          = -
  vendor_os_id = 7 (suse-10.3-main)

x60s:~ # slxconfig change-system suse-11.0-clone::nfs \
scratch="nfs://10.8.4.2/data/export/tmp"
```

## System-Update and Extension

With Stage1-Installations installed directly from the package source, you perform updates on the server. This is triggered by the "update" parameter in slxos-setup:

```
x60s:~ # slxos-setup update suse-11.0-kde
Setting up Update Process
Setting up repositories
base_non-oss      100% |=====| 951 B 00:00
base              100% |=====| 951 B 00:00
base_update      100% |=====| 1.2 kB 00:00
Reading repository metadata in from local files
primary.xml.gz   100% |=====| 548 kB 00:00
##### 1340/1340
Resolving Dependencies
[ ... etliche weitere Ausgaben ... ]
Starting SuSEconfig, the SuSE Configuration Tool...
Running in full featured mode.
Reading /etc/sysconfig and updating the system...
Executing /sbin/conf.d/SuSEconfig.perl...
Executing /sbin/conf.d/SuSEconfig.permissions...
Finished.
Vendor-OS 'suse-11.0-kde' updated succesfully.
```

The update of a clone system is performed on the reference machine in the so called Stage0. After this you call  
"slxos-setup clone 11.00.4.50:/ suse-10.3-mysystem":

```
Vendor-OS 'suse-10.3-mysystem' has been re-cloned successfully.
No need to change vendor-OS 'suse-10.3-mysystem' in OpenSLX-database.
```

```
x60s:~ # slxos-setup clone 0.20.4.50:/ suse-10.3-mysystem
building file list ...
[ ... etliche weitere Ausgaben ... ]
sent 434920 bytes received 121491994 bytes 375737.79 bytes/sec
total size is 8644862874 speedup is 70.90
Vendor-OS 'suse-10.3-mysystem' has been cloned successfully.
No need to change vendor-OS 'suse-10.3-mysystem' in OpenSLX-database.
```

You will then be informed that `slxos-setup` was completed successfully. There are no special update parameters for System Clone since the update occurs externally to the server on the reference machine and is only then copied to the server.

## Cleanup

At some stage you may want to remove older systems or test versions. The tools `slxos-setup` and `slxos-export` have a delete operation for this. For this you work in reverse order. `slxconfig` deals with the database and doesn't delete anything in the file system:

```
x60s:~ # slxconfig remove suse-11.0
removing vendor-OS folder '/data/openslx/stage1/suse-11.0'...
Vendor-OS 'suse-11.0' removed successfully.
Vendor-OS 'suse-11.0' has been removed from DB!
```

The removal of systems from the `~/tftpboot` area occurs with a recall of `slxconfig-demuxer`.

In the next step you remove exports (parts of Stage2), which do not use an insignificant amount of disk space. This process also removes the associated database entry.

**% change!!**

```
x60s:~ # slxos-export remove suse-11.0 nbd
export '/srv/openslx/export/nbd-squash/suse-11.0' successfully removed!
Export 'suse-11.0' has been removed from DB.
```

```
x60s:~ # slxos-export remove suse-11.0
export '/srv/openslx/export/nfs/suse-11.0' successfully removed!
Export 'suse-11.0' has been removed from DB.
```

In the last step we remove the installed base system (Stage1) from the disk of your server. This also removes the database entry.

```
x60s:~ # slxos-setup remove suse-11.0
Vendor-OS 'suse-11.0' removed successfully.
Vendor-OS 'suse-11.0' has been removed from DB!
```

## Boot Cycle

### PXE-Boot

Your PCs can now do what, for a long time, was a given on a "proper" Unix Workstation: Intels Pre boot eXtension Environment (PXE) gives the option of booting hard drive-less machines over a network. The network booting machine acquires an IP Address via DHCP as well as the information of where it can load a PXE-Image via TFTP. This then gives the machine more boot functionality. At this point a whole class of network software is added. Well known are the wide range of netbased operating system installations, whether Linux or Windows.

The PXE Boot menus for you clients are prepared by the `slxconfig-demuxer` automatically in the directory `~/tftpboot/pxe/pxelinux.cfg`. If you would like to customize the look of your menu or change the default password for access to the appropriate kernel command line, you must edit the file `/etc/opt/openslx/PXE-template`.

## DHCP for the IP-Address assignment

If there was not already a DHCP server in your network, you will now need to setup one. In most cases it is sufficient to adjust two files to get a DHCP server running. The entries for your clients are found in `/etc/dhcpd.conf`. For initial test the following entries are enough:

```
ddns-update-style none;
subnet 10.8.4.0 netmask 255.255.255.0 {
    server-identifier 10.8.4.1;
    next-server 10.8.4.1;
    filename "pxe/pxelinux.0";
    option routers 10.8.4.254;
    range 10.8.4.101 10.8.4.200;
}
```

After this you determine in `/etc/sysconfig/dhcpd` that the DHCP server is listening on the correct interface and, for security, is running in a chroot environment:

```
DHCPD_INTERFACE="eth0"
DHCPD_RUN_CHROOTED="yes"
```

## TFTP

The start procedure of a client upto and including Stage3, is strongly associated with TFTP. The initial data (PXE-Linux and the menu system) and the client configuration are acquired using this protocol. The directory structure for a typical SLX-Installation looks like this:

```
tftpboot/client-config
tftpboot/client-config/suse-11.0-A
tftpboot/client-config/suse-11.0-A/01-<MAC>.tgz
tftpboot/client-config/suse-11.0-A/default.tgz
tftpboot/client-config/<system>/...
tftpboot/pxe
tftpboot/pxe/menu.c32
tftpboot/pxe/pxelinux.0
tftpboot/pxe/suse-11.0
tftpboot/pxe/suse-11.0/initramfs-1
tftpboot/pxe/suse-11.0/vmlinuz
tftpboot/pxe/<vendor-os>/...
tftpboot/pxe/...
```

By default, the directory is located under `/srv/openslx`. Its location can be changed by using `slxsettings --tftpboot-path=New-Dir`.

The file `in.tftp` is included with most distributions as TFTP server. This is activated using `xinetd`. For this you edit the file `/etc/xinetd.d/tftp` depending on the distribution:

```
service tftp
{
    socket_type          = dgram
    protocol             = udp
    wait                 = yes
    user                 = root
    server                = /usr/sbin/in.tftpd
    server_args          = -s /tftpboot
    disable              = yes
}
```

The carried out settings for the directory need to match the settings of the OpenSLX package and DHCP server. The variable "filename" contains `"/tftpboot/pxe/pxelinux.0"`, this file should be delivered to the

requesting PXE-Clients.

The entry "-s /tftpboot" means, that at the start of the in.tftpd, under the entered path, clients can download files from the server. So that the client knows where to search, the DHCP server informs them with the variable "filename". PXE itself is not in the position to load a large kernel which may be a few megabytes straight away. So, it is given a special Boot-Loader from the Syslinux package from Peter Anwin, pxelinux.0.

This tool takes over all further steps and can also build a boot menu and provide further boot options. After the establishment you restart xinetd, you then use the small command tftp from any machine in your network or directly from the server to check if the file pxelinux.0 can be loaded from the location your clients are expecting:

```
dirk@x60s:/tmp> tftp 10.8.4.254
tftp> get /tftpboot/pxe/pxelinux.0
tftp> quit
dirk@x60s:/tmp> ls -la pxelinux.0
-rw-r--r-- 1 dirk users 13320 2007-04-20 16:10 pxelinux.0
```

## Syslinux

In the last section we discussed a program to produce boot menus and to load the kernel and InitialRamFS through TFTP: pxelinux.0 is a component of the Syslinux package. The central program pxelinux.0 and menu-help program are available in the OpenSLX package.

There are some specific parameters for PXELinux. pxelinux.0 expects its configuration file in respect to some locations under pxelinux.cfg. If no special machine specific configuration files are found, it loads a file named default. Machine specific files have the format 01-<MAC-des-Clients>. The format of specific and default files is identical, their content may of course not differ. In your system settings you will find the template:

```
NOESCAPE 0
PROMPT 0
TIMEOUT 10
DEFAULT menu.c32
IMPLICIT 1
ALLOPTIONS 1
MENU TITLE What would you like to do (Selection using cursor)?
MENU MASTER PASSWD secret
LABEL memtest
    KERNEL memtest.bin
```

It is best to use the small test tool memtest for the first test of a PXE-Boot over the network. It is a small binary program which performs intensive memory tests in infinite test runs. So that you don't always need to get a CD with the current version, you can easily provide it through the network for all PXE compatible machines. The tool can be found, for example, on a magazine DVD or the first installation CD of SuSE and should be copied into the same directory as pxelinux.0. If it is in a different location, you must ensure that the TFTP server has access to this location. You can also specify the full path like "kernel /tftpboot/tools/memtest.bin". The termination of memtest can be chosen arbitrarily, but may not be "\*.0". If everything worked well, you can then experiment using a Linux Kernel and an appropriate Ramdisk in the next step.

```
prompt 1
timeout 100
default dxs
label dxs
    kernel dxs
    append vga=normal initrd=initrd-dxs nfsserver=10.30.4.1:/nfsroot/ldc
    ipappend 1
label memtest
    kernel memtest.bin
```

Most of the parameters are self explanatory: Prompt ensures that PXELinux with the prompt "boot:" waits for 100 seconds (timeout 100). If in this time no input was given, by default "label dxs" will be started. The setting assumes the kernel called kernel-dxs and a matching Ramdisk called initrd-dxs is located in the directory /nfsroot/ldc/boot. This can easily be achieved with:

```
cp /boot/vmlinuz /nfsroot/ldc/boot/kernel-dxs
cp /boot/initrd(.img) /nfsroot/ldc/boot/initrd-dxs
```

"ipappend 1" ensures that PXE writes the configuration in the Kernel-Commandline which is acquired through DHCP. This contains the additional parameter "ip":

```
ip=11.00.40.2:11.00.40.71:11.00.40.254:255.255.255.0
```

You can view the content of this special row with the command "cat /proc/cmdline". After the configuration has been completed, you will see a lot more with a restart of the Test-Clients. Now the client almost looks like usual: A boot loader takes care of the Kernel as well as the Initial Ramdisk and ensures they start. The Kernel extracts itself, as is familiar with Knoppix or a hard drive installation. The Mini-Linux-Environment of the Initial-Ramdisk is carried out. However the process breaks with a failed mount of the Root-filesystem.

Image: vmware-normalinitrd.png

This is because the Kernel and Ramdisk come from a classic installation and are prepared for a start from a hard drive. At this point the Admin has come quite far and has enough of an understanding of the basic concepts to try out a Diskless Linux project.

## Configuration of OpenSLX Clients

OpenSLX clients cannot store their configuration as is the case with classic Linux-Workstations. This is why machines are prepared in multiple steps:

1. You acquire your Base-IP-configuration when booting through PXE or Etherboot/gPXE if in the PXE file the Parameter IPAPPEND 1 is set. Otherwise the machine will call a new DHCP-Client in Stage3.
2. If the Kernel started in the Stage3-Init, the main script will read the configuration data from the previous stages Kernel command line and the InitRamFS delivered machine setup file.
3. After this an extended configuration within Stage3, depending on the specified configuration start, is available.
4. Then the Stage3 scripts read the loaded configuration and create the majority of the settings for Stage4.

All primary configuration steps occur during the run of InitialRamFS. The central configuration file is machine-setup. It is populated and extended during the setup of InitialRamFS in Stage2 initialization and during Stage3. In Stage4 it can be accessed in the directory /etc.

## The Kernel Command line

The slxconfig-demuxer prepares the Kernel Command line alongside the InitialRamFS and Client configuration. It consists of necessary and optional entries which are delivered by PXELinux or Etherboot. The IP configuration can and InitialRamFS must count towards this. However there is a problem, the Kernel command line with Intel platforms is limited to 256 characters, so must take this into account for this entry.

- debug - without specifying a level, the debug level is set to 1
- debug=\* - sets the debug level on the specified integer value. For a listing of all debug levels see appendix E.2

- `nodhcp` - no renewed DHCP for configuration in `InitRamFS`. Setting for cases where the standard IP and DNS data are assigned in other ways (`ConfTGZ` and `PXE-IP-Info`). This entry is not in the Kernel command line by default.
- `ldap=*` - use LDAP for configuration (URI Syntax)
- `file` - use TFTP to acquired configuration files
- `file=*` - use TFTP take the specified TFTP Server and path (URI Syntax)
- `ldsc` - do not generate `ld.so.cache` and overtaken by existing3 from Stage2.
- `rootfs=*` - Type of Root file system of the Client. In question are currently NFS, (D)NBD with SquashFS. The URI Format is expected, for example "`rootfs=nfs://server/pfad`" or "`rootfs=nbd://server:port/Filesystem`". If the root filesystem is made up of many components, "`ldsc`" will automatically be activated. Instead of a constant server IP, you can use the variable "`@@serverip@@`" when DHCP- und Root file system servers are identical. In Stage3 it will be replaced with the DHCP acquired value for the DHCP server.
- `dcsize=*` - only needed with defined "`rootfs=dnb://...`", defines the size of the cache file for DNBD. Since the cache file is stored in RAM, it should be chosen according to available RAM.
- `ip=*` - is populated through PXE-Linux or Etherboot/gPXE.
- `tmpfssize=*` - size of the maximum temporary memory in RAM (`tmpfs`)
- `vci=*` - Vendor Code Identifier, only needed if DHCP is used.

## Central Configuration File machine-setup

The central Configuration File of a OpenSLX client is `machine-setup`. In the current version it can be found in `/var/opt/openslx/config/default/initramfs/machine-setup`. For a given installation it should already be populated with reasonable settings, since it is entered into the created `InitRamFS` as `Initial-Configuration4`. The configuration methods in the attached explanation, append their results to this file. So for example, the DHCP configuration could possibly overwrite the variable for `Hostname`, `DNS-Server`, ... etc. In some cases a variable can be defined more than once. In this case only the last one defined is valid, so the file is read from the bottom up if you are looking for a particular assignment. Instead of a constant Server IP for the differing sources or services, you can use the variable "`@@serverip@@`" in `machine-setup`. It will be replaced with the address acquired from DHCP in Stage3. With this you can spare yourself statistical entries if the same IP address is valid for most or even all SLX-Services.

Even though in most environments, OpenSLX Clients are kept quite consistent, it could be of interest or necessary to specify settings for individual clients separately. This becomes more important in larger networks with groups of clients. All of the following configuration types are accessed in Stage3 of the client start.

Currently there are three (four) configuration possibilities, even if they aren't all implemented:

Via TFTP (configuration plus extension - `ConfTGZ`)

`file get` - is a component of the current version 4. Extension to database generated `machine-setup`, like user authentication, home directory and admin specific scripts (summarized as `TGZ`) is obtained by the client (MAC sei `00:11:43:7c:da:ff`) one after the other in the Form `01-00-11-43-7c-da-ff.tgz` and `default.tgz`.

The files are obtained at the start of the `Services-Configuration` (`servconfig`), so could possibly be modified.

Files can be specified explicitly:

```
"file=tftp://<server-ip>/pxe/client-config/<system-name>/01-00-11-43-7c-da-ff.tgz"
```

or should be present under a certain schema.

For this reason, if problems arise, you should test the client first to see if the command line is populated with all options.

```
apt-get install libdbd-sqlite3-perl sqlite3
```

## Problems

```
slxconfig-demuxer --verbose-level=2
```

```
slxos-setup remove suse-11.0-x86_64
```

## OpenSLX - Step-By-Step-Howto - OpenSLX Lab

```
removing vendor-OS folder '/var/opt/openslx/stage1/suse-11.0-x86_64'...  
Vendor-OS 'suse-11.0-x86_64' removed successfully.
```

## Further Development

With technologies like UnionFS or AUFS, an existing base installation can easily be extended or cut back. This allows different user groups to be served separately. Furthermore it is possible to setup a central server that allocates ground installation/infrastructure, and further software is integrated into the system locally.

Completely different possibilities are offered through virtual workstations: Instead of installing the OS for a certain user directly, this occurs in a virtual environment, which places the container in the user's home directory. So, users can access any machine in the OpenSLX network and will find their familiar environment.

Also available in: [HTML](#) [TXT](#)

Loading...

Powered by [Redmine](#) © 2006-2009 Jean-Philippe Lang