NAIS-Documentation-HOWTO

Mattias Gärtner *gaertner@informatik.uni-koeln.de*, Lech Nieroda *nieroda@informatik.uni-koeln.de*, Jens Rühmkorf *ruehmkorf@informatik.uni-koeln.de*.
v.0.1, June 2000

This document describes how to automatically install a PC Cluster with Linux via network using NAIS. It is a step-by-step guide that explains the set-up of the needed services on your server and what part on the client side is required.

Contents

1 Introduction

For people who are involved in administrating networks of Linux computers installing or configuring computers is an everyday task. Whenever manual interaction is involved to set up and configure a computer, it is likely that you miss something which will lead to mistakes. Moreover, it is rather boring to perform the same task again and again. Thus it is clear that a system that lets you perform a non-interactive, network based automatic installation will surely save you a lot of time in the long run. That is what NAIS lets you do for ix86-architectures.

1.1 Disclaimer and Copyright

1.2 Intended Audience and Applicability

Most Linux users should never have to even look at this document because many Linux distributions do a pretty good job at aiding the user to set up and configure his system. The information in this document is aimed at users who have experience with administrating Linux machines and wish to set up a network

automatic installation system using Debian GNU/Linux as distribution. Everything is ONLY tested to work for ix86-architectures.

1.3 Features

Just a short list of features NAIS has we thought an automatic installation system should have. Feel free to mail us if you miss something:

- The Linux distribution you use for your server does not need to be Debian GNU/Linux or even potato; we have set up a fully functional server system on SuSE 6.3 as well as on Debian GNU/Linux 2.1 using the same scripts.
- You can use the same kernel for installation that you use for running the client. The kernel just needs ramdisk- and initrd-support compiled in as well as the network device support to work.
- The use of initrd makes it possible to automatically install clients with "unusual" network interfaces, such as PCMCIA, whose support cannot be compiled into the kernel yet. NOTE: The `initrd.gz` we use does not support PCMCIA as it is, you have to add the needed binaries and modules yourself to make it work.
- There is no second reboot required – pretty much the same like some other distributions do it.
- Uses a convenient "class" concept for configuring.
- The tool `setup_harddisks` lets you specify general rules for partitioning and formatting the client's local hard disks; preserving already created partitions is possible as well.
- All parameters are passed to the client using DHCP.
- We took care to keep everything RFC-compliant so you should be able to use etherboot, netboot, pxelinux or BpBatch to boot over network.
- A `sysinfo` mode that just gathers some significant information about the client and gives you a shell (remember, `telnet client 2323` instantly reboots the client, so you can care less about the shell). Other modes possible but not implemented yet.
- The `sysinfo` and `install` mode let you establish a secure terminal session to the "install-client" using ssh (once `sshd` is up and running). Idea from Thomas Gebhardt.
- You can request your client via tcp to reboot, at any time, in case the installation fails. Gets quite handy when the client you want to install is 400 meters away ;)

If you just started reading this document, some of these features might not be clear to you. They will be explained in detail in the following sections.

1.4 Wishlist

Just some features we thought an automatic installation system should have which we did not have the time or resources to implement yet.

- A `backup` mode.
- Better debconf support. So far we only use debconf's feature to run apt in "non-interactive" mode.
- Adapt `setup_harddisks` to work with reiserfs.

- Make a debian package of NAIS.
- We did not have time to add support for etherboot yet, only netboot works as well as pxelinux and BpBatch.
- Not concerning NAIS but still important: There happens to be no debian package for etherboot yet (but for netboot there is one). Hmm. Time to become a debian package maintainer.
- Make an anonymous (pserver) CVS repository available for those who want to have the latest and greatest ;)

1.5 New versions of NAIS and this document

You can always find the latest version of NAIS as well as the most recent version of this document on the World Wide Web via the URL `<http://www.informatik.uni-koeln.de/nais/>`.

1.6 Feedback and corrections

If you have questions or comments about this document, please feel free to mail us at *nais@informatik.uni-koeln.de*. We welcome any suggestions or criticisms. If you find a mistake within this document or experience any problems using NAIS, please let us know so we can correct it in the next version. Thanks.

2 Quick-Installation-Guide

Not yet, sorry. But all necessary information can be found in the other sections. Just no quick guide for the impatient :(

3 Setting up the server

The server works as a repository. The clients will need to access server ressources in order to boot, install packages and so on. To achieve this, various network services must be properly set up.

Most files required for the installation are located under `/files/install`. The `nais` directory contains all configuration files as well as scripts and required utilities.

```
vermeer[~]# tree -d -L 2 /files/install/nais/

/files/install/nais/
|-- config
|   |-- class
|   |-- disk
|   |-- env
|   |-- files
|   |-- package
|   -- scripts
|-- install
|   |-- init.d
|   |-- rc0.d
|   -- rc1.d
-- utilities
    |-- busybox
```

```
|-- cmos
|-- install_packages
|-- kiss
|-- mk_initrd
|-- mk_mbaimg
|-- mk_root
`-- setup_harddisks
```

The `installroot` directory contains a root filesystem based on Debian's base2_2.tgz. It will be mounted by the clients during installation, providing a basis for their respective filesystems.

```
vermeer[~]# tree -d -L 2 /files/install/installroot/

/files/install/installroot
|-- bin
|-- boot
|-- busybox
|-- cdrom
|-- debian
|-- dev
|-- etc
|-- floppy
|-- home
|-- initrd
|-- kiss
|-- lib
|-- mnt
|-- proc
|-- root
|-- sbin
|-- tmp
|-- usr
-- var
```

Now we can set up the network daemons.

3.1 The TFTP daemon

A client should boot from its network card or floppy. In order to boot via the network card, the TFTP service must be installed on the server.

A first step is to add or uncomment the following lines from /etc/inetd.conf:

```
tftp dgram udp wait.240 nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /tftpboot
```

Note: the directory /tftpboot will be used as a repository for the kernel and the ramdisk image the client will boot from After changing this file, inetd must update its configuration. To achieve this, it must be killed and restarted (the HUP option):

```
vermeer[~]# killall -v -HUP inetd
Killed inetd(15086)
```

In case your linux distribution doesn't have the command `killall`, which kills processes by name, you can use `kill` with the appropriate process id as well.

3.2 The NFS daemon

The clients access all server resources via NFS. To enable NFS service, rcp.nfsd and rpc.mountd daemons must be started. Debian achieves this by executing the nfs-server script, which is located in `/etc/init.d`. The clients are only able to mount exported directories. Alter the /etc/exports accordingly :

```
vermeer[~]# cat /etc/exports
/files/install              134.95.10.128/255.255.255.128(ro,no_root_squash)
/debian                     134.95.10.128/255.255.255.128(ro,root_squash)
```

3.3 The DHCP daemon

The main function of the DHCP server is to give the client an IP adress and to make it load the file named bpbatch.P from the TFTP server. Note: DHCP is a superprotocol over BOOTP. The installation and configuration of DHCP has been well documented in the `<http://linuxdoc.org/HOWTO/mini/DHCP/>` . Refer to it for details.

3.3.1 Preliminaries

First of all, you should check whether your kernel does have MULTICAST support enabled. You can do so by typing:

```
vermeer[~]# ifconfig -a
.
.
eth0    Link encap:Ethernet  HWaddr 00:50:DA:41:E0:1C
        inet addr:134.95.10.140  Bcast:134.95.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

On most systems this option is already set.

DHCPd stores information about current leases in a file named dhcpd.leases. It is in plain text format, so you can view it during DHCPds operations. You will propably have to create it; type:

```
touch /var/state/dhcp/dhcpd.leases
```

Older versions of dhcpd (<2.0) store this file as `/etc/dhcpd.leases`

3.3.2 Configuring DHCPd

All options for the DHCP deamon can be set in the `/etc/dhcpd.conf` file. If you are using KDE 2.0 you can try `kcmdhcpd` - a graphical interface for dhcpd configuration, very similar to the Windows NT DHCP configurator.

If you are using a PXE boot-PROM, add the following lines :

```
option dhcp-class-identifier "PXEClient"
option vendor-encapsulated-options ff;
```

The dchpd.conf consists of two main parts.

- global options
- bootmethod specific options

Setting the global options Here is an excerpt of our `dhcpd.conf`, containing the global options. All option tags (e.g. `option-211`) have corresponding environment variables (e.g. `${NAIS_ACTION}`). Adjust them to your needs as you see fit. Note: the global parameters can be overwritten by specific options if needed; all comments are preceded by a `#`.

```
# ${NAIS_ACTION}
# NAIS can do more than installing a client :) Possible options:
#
#   install -- install the system
#   sysinfo -- fetch all hardware information and write it to logs
#   bash -- start a bash
option option-211 "bash";

# ${NAIS_PATH}
# NAIS will be mounted during installation via nfs
# from the dhcp server (see statement "server-name" below).
# This is its directory.
option option-212 "/files/install/nais";

# ${NAIS_ROOTPATH}
# After successful booting and extracting the ramdisk
# containing the miniroot system, NAIS mounts this
# directory from the dhcp server. The needed binaries,
# libraries and such are taken from this directory;
# it is actually a debian base with some additional
# packages.
option option-213 "/files/install/installroot";

# ${NAIS_TFTPLINK}
# After installation NAIS switches the boot method (e.g.
# to boot from harddisk. To do this easily the bootfile
# is only a link which will be switched after installation.
option option-214 "hdboot.bpb";

# ${NAIS_BOOTMETHOD}
# The boot method. Current values are "bpbatch", "pxelinux",
# "netboot" and "etherboot".
option option-215 "bpbatch";

# ${NAIS_FLAGS}
# Some flags for the general behaviour of NAIS
# "verbose"     increase output
# "debug"       write every line of code
# "keep_bootmethod"     normally the client changes the boot
#               method after the installation to boot from its
#               harddisk. This flag deactivates this feature so
#               that every time the client reboots it will be
#               reinstalled.
# "no_reboot_daemon"    do not start the reboot daemon "kiss".
# "reboot"      normally NAIS installs without rebooting. With this flag
#               the client reboots after installation.
# "paranoid"    interactive mode, for those who don't have trust
#               in our **automatic** installation ;)
# NOTE: Obviously combinations like "reboot" and "keep_bootmethod"
# don't make much sense. Nevertheless they are possible.
```

```
option option-216 "keep_bootmethod verbose";

# ${NAIS_USER}: account on tftp and log-server
# Used to save all log-files to server and to change image which is
# booted via network. For this to work you need to configure this user's
# .rhost, so root can login from all install clients without password.
# As well, this account must have write permissions for /tftpboot.
#
# Note:  To achieve this we granted write permissions for /tftpboot to
# the group linuxadm, which NAIS_USER is member of.
# chgrp linuxadm /tftpboot; chmod g+w /tftpboot
option option-217 "nais";

# ${NAIS_LOGSERVER}
# NAIS saves at the end of every phase all logfiles to this server
# using the account NAIS_USER via rsh/rcp.
option option-218 "134.95.10.140";

# ${NAIS_BINSERVER
# Server from which to mount bin-trees like /usr etc. Only useful
# for lean-clients. To use it, you have to define a class USR_MOUNT
# or USR_LOCAL_MOUNT.
option option-219 "134.95.10.140";

# ${NAIS_HOMESERVER}
# Server from which to mount /home. Only useful if you define a
# class named HOME_CLIENT
option option-220 "134.95.10.140";

# ${DEBIAN_NFS_LOCATION}
# Packages are installed with APT and debconf. APT can fetch
# the debian packages via nfs, http or ftp. If some of your
# client use nfs you must tell NAIS the path.
option option-221 "134.95.10.140:/debian";

# ${NAIS_SEARCH}
# Additional values for search list in resolv.conf.
# E.g. if you want your resolv.conf to look like
# -- snip --
# search informatik.uni-koeln.de uni-koeln.de
# -- snap --
# you have to define this to be "uni-koeln.de".
# Otherwise, leave it empty.
option option-222 "uni-koeln.de";

allow booting;
allow bootp;
deny unknown-clients;
default-lease-time 60;
max-lease-time 70;

#
# It is important to get hostnames as well (not only ip)
# because we let NAIS use hostnames instead of ip adresses.
#
```

```
use-host-decl-names on;
```

The bootp flag (`allow bootp;`) tells dhcpd to respond to bootp queries.

These options are global but specific to our network:

```
server-name "134.95.10.140";
option domain-name "informatik.uni-koeln.de";
option domain-name-servers 134.95.100.209, 134.95.100.208, 134.95.140.208;
option time-servers vermeer;
option ntp-servers 134.95.99.254, 134.95.110.33, 134.95.111.13;
option nis-domain "pinguine";
option nis-servers vermeer;
option subnet-mask 255.255.255.0;

subnet 134.95.10.0 netmask 255.255.255.0 {
  option routers 134.95.10.254;
  }
```

The server will advise the client to use 255.255.255.0 as its subnet mask, 134.95.10.254 as the router/gateway and 134.95.100.209, 134.95.100.208, 134.95.140.208 as its DNS servers.

Setting the bootmethod specific options Decision time. Now you can choose from the following bootmethods:

- PXElinux
- BpBatch
- Netboot
- Etherboot (caution: this option has not been tested yet)

This is a sample configuration of a client using PXElinux.

```
option option-215 "pxelinux";         # NAIS_BOOTMETHOD
option vendor-encapsulated-options 09:0f:80:00:0c:4e:65:74:77:6f:72:6b:20:62
 :6f:6f:74:0a:07:00:50:72:6f:6d:70:74:06:01:02:08:03:80:00:00:47:04:80:00:00
 :47:04:80:00:00:00:ff;
filename "/tftpboot/pxelinux.bin";
option option-211 "install";          # NAIS_ACTION

host roy00 {
  hardware ethernet 00:10:5A:25:DB:BA;
  fixed-address 134.95.10.160;
  option option-214 "865F0AA0";       # NAIS_TFTPLINK
}
```

This will result in DHCP server giving a client (roy00) a specific IP adress (134.95.10.160) based on its ethernet adress (00:10:5A:25:DB:BA).

You can mix these methods, like assigning 'static' IP adresses to certain clients and dynamic IPs to mobile users (laptops). For further options refer to the *dhcpd.conf(5)* man page.

When using BpBatch, you will have to create appropriate configuration files. They are usually different for each client and describe which kernel and ramdisk will be loaded. Note: these files do have a `.bpb` extension, but only their basename has to be written here.

```
option option-215 "bpbatch";          # NAIS_BOOTMETHOD
option option-214 "hdboot.bpb";       # NAIS_TFTPLINK
filename "/tftpboot/bpbatch";
option option-211 "install";          # NAIS_ACTION
host roy00 {
    hardware ethernet 00:10:5A:25:DB:BA;
    fixed-address 134.95.10.160;
    option option-135 "/tftpboot/roy00"; # bpbatch config file (roy00.bpb)
}
```

The netboot method doesn't differ much from the preceding one:

```
option option-215 "netboot";          # NAIS_BOOTMETHOD
always-reply-rfc1048 on;
option option-211 "install";          # NAIS_ACTION
option option-129 "reboot=warm";      # kernel parameters
host roy00 {
    hardware ethernet 00:10:5A:25:DB:BA;
    fixed-address 134.95.10.160;
    filename "/tftpboot/roy00";
}
```

Etherboot. This method has not been tested or even implemented yet, proceed with caution.

```
option option-215 "etherboot";        # NAIS_BOOTMETHOD
option option-211 "install";          # NAIS_ACTION
host roy00 {
    hardware ethernet 00:10:5A:25:DB:BA;
    fixed-address 134.95.10.160;
    filename "/tftpboot/roy00";
}
```

3.3.3 Starting the daemon

In order to start the DHCP server just type

```
/usr/sbin/dhcpd
```

When running dhcpd for the first time, you might want to use the options `-d -f` to enable debugging messages and put them on the server console. Boot your client and check the result. If everything works out fine you have made it.

4 Setting up the client's resources

This section explains how the client's resources are set up on your server. First of all, an install-client is supposed to boot from its network card, thus the DHCP, NFS, and TFTP services should be installed and properly set up on your server(s), refer to section 3 (Setting up the server) for details. Although the installation is based on the Debian GNU/Linux "potato" release, it can be employed on a server running different linux distributions as well; we have set up a fully functional server system on SuSE 6.3 as well as on Debian GNU/Linux 2.1 using the same scripts.

To give you a brief overview of the installation process:

1. The client boots via the network (see section 5 (How the client boots) for details). Then, the net-boot loader loads the kernel `bzImage` and the initial RAM disk image `initrd.gz`. (By changing this `initrd.gz` additional modules can be loaded as well e.g. to enable PCMCIA support.)

2. The ash-script `/linuxrc` is executed (in fact, this can be any valid executable; it is run with uid 0 and can do basically everything init can do):

   - The dhclient tries to configure the network interfaces that are configured in the system and asks for additional options (especially vendor tags) using DHCP protocol.
   - Our "reboot daemon" `kiss` is started, usually listening on port 2323. Thus, a `telnet client 2323` will **imediately** reboot the client (unless the installation has already finished).
   - The binaries and libraries, `server:/some_path/installroot/`, as well as NAIS' base directory, `server:/some_path/nais/`, with further scripts and configuration files are mounted readonly from the server.
   - Theoretically `linuxrc` is still running but practically it forks a bash-script via NFS from the server to do the rest. Everything else is now beyond the scope of `linuxrc`.
   - The client's hardware (especially its hard disk) is set up and a Debian GNU/Linux base system is installed and configured.

3. As `linuxrc` terminates, the "real" root file system (e.g. `/dev/hda1/`) is mounted and the 2nd part of the installation begins: The left packages are installed and configured.

4. `kiss` is stopped.

5. Installation is completed!

4.1 How to create a reasonable client kernel

The kernel compilation procedure is documented extensively by various documents; in case of trouble you might want to refer to *Kernel-HOWTO* `<http://linuxdoc.org/HOWTO/Kernel-HOWTO.html>` or the kernel's `Documentation/`-directory for example.

For building a linux kernel suitable for installing and running a client choose appropriate options and drivers that match your client's hardware configuration (e.g. hard disks, cdrom etc.). Note: if you can avoid using modules to support your clients hardware devices, do so - that will simplify your further tasks; you will need them for PCMCIA though.

Furthermore, the kernel needs following features built-in:

- Network device support (`CONFIG_NETDEVICES=y`) for the clients NIC.
- NFS filesystem support.
- Initial ramdisk (and therefore ramdisk) support.
- Packet protocol (`CONFIG_PACKET=y`) and Linux socket filter (`CONFIG_FILTER=y`).
- Any other low-level drivers necessary to detect and use the client's hardware.

If you think you missed something take a look at the kernel-config files that come with NAIS. Also, when compiling the modules remember to set the environment variable `${INSTALL_MOD_PATH}`:

```
vermeer[/usr/src/linux]# setenv INSTALL_MOD_PATH /tmp
vermeer[/usr/src/linux]# make modules modules_install
.
.
Installing modules under /tmp/lib/modules/2.2.16/block
Installing modules under /tmp/lib/modules/2.2.16/fs
Installing modules under /tmp/lib/modules/2.2.16/misc
```

4.2 How to create install-root filesystem

The install-root filesystem is based on Debian's base2_2.tgz. Propably, it will have to be made only once. It will be mounted read only by the clients during the installation, thus being the basis of their respective filesystems. It is quite easy to create the root filesystem with Debian.

4.2.1 Customizing util.conf and mk_root

Edit the two files util.conf and mk_root in this directory. Both contain numerous comments which should explain in detail the options you are asked to set. You should look through util.conf and mk_root and check the variable settings.

util.conf

This file contains basic informations about your server and various (remote and local) file locations. Review the PACKAGES variable carefully - it enumerates the packages to be installed in the installroot. Make sure you specify the INSTALL_ROOT - that is where your installroot will be created

mk_root

This script specifies how your installroot will be built and what it will contain.

4.2.2 Making the installroot filesystem

Make sure you don't run the script from your root directory ("/"). In case this happens by accident, the script will exit immediately. The script retrieves the files base2_2.tgz and drivers.tgz from the internet, if needed, and extracts the debian "base" root-filesystem as well as the modconf package from drivers.tgz :

```
case $BASE_TGZ in
  ftp:*|http:*)
    file=$INSTALL_ROOT/${BASE_TGZ//*\//}
    info "Retrieving baseX_Y.tgz ..."
    wget -nv -P$INSTALL_ROOT/ $BASE_TGZ
    info "Unpacking baseX_Y.tgz ..."
    tar -C $INSTALL_ROOT -zxpf $file
    rm -f $file
  ;;

  *)
    info "Unpacking baseX_Y.tgz ..."
    tar -C $INSTALL_ROOT -zxpf $BASE_TGZ
  ;;
esac
...
tar zxOf $DRIVERS_TGZ ./modconf.tgz | tar -C $INSTALL_ROOT -xpz
```

...

Some scripts needed later on (like chroot_script), busybox sources, an apt-get configuration file (URL's of available debian ftp-servers) and the resolv.conf are copied into the install-root. In order to avoid needless modules probing, the switch_off file, containing the neccessary 'off' lines for the disk-modules, is copied to the etc/modutils directory. The modules.conf will be updated in the chroot_root.sh script. As a last step, we link etc/mtab and etc/fstab to proc/mounts. This is neccessary, since these files are required for unmounting. Note: proc/mounts is not being mounted in the chroot environment, but it will be during the installation (linuxrc script).

4.2.3 Installing additional packages

Now that the preliminaries have been dealt with, we can change the root directory to the install_root. This step, made incorrectly, can damage your system. Thus, we decided to let the script perform this operation :

```
chroot $INSTALL_ROOT /bin/bash ${__chroot//*\//}
```

Once inside the install-root, the chroot_script.sh is executed. Note: you are in a debian system now - all operations made herein are independent of your local linux distribution. For the next step you will need internet access. As an alternative, a local (NFS) debian mirror can be mounted at "/debian".You can set this option in the util.conf file.

```
[ x$APT_METHOD = xfile ] &&
  mount -n -t nfs -o ro $NFS_SERVER /debian
```

Apt-get will download and install the newest versions of some needed packages. In detail, we need :

- for compiling: make, g++
- for the miniroot: loadlin (for freeramdisk), ash
- for the installroot: cfengine, file, perl-5.005, perl-5.005-debug, less, bootpc, rdate, dnsutils, strace (debug)

Note: apt-get will show you all dependencies of these packages and ask for approval, so you can consider a change in the PACKAGES variable. After the successful upgrade/installation of these packages, busybox will be compiled. In order to avoid needless modules probing, we update the modules.conf file and create an empty modules.dep file.

If everything went well, your install-root should be complete.

4.3 How to create the Initial RAM disk

Note: The initrd is independent of your linux distribution, since all needed files, in particular libraries, are copied from the Installroot. If you have not created it by now, you'd better return to the section 4.2 (How to create install-root filesystem).

4.3.1 ... and why do we need it ?

The initial RAM disk contains a minimal root filesystem. It is used to mount the neccessary ressources from the server. Furthermore, once the client is installed, it does not have to be rebooted - the root simply changes from the initrd to the created one. Apart from the installation, it can be used conveniently as a boot- or rescue-disk. Originally, the initrd has been designed to load additional modules during system startup, thus making pcmcia-usage possible. Look out for this feature in a future release.

4.3.2 Customizing util.conf and mk_initrd

Edit the two files util.conf and mk_initrd in this directory. Both contain numerous comments which should explain in detail the options you are asked to set. You should look through util.conf and mk_initrd and check the variable settings.

util.conf

This file contains basic informations about your server and various file locations.

mk_initrd

This script specifies how your initial RAM disk will be built and what it will contain. In case you want to use your initrd for other purposes than this installation, you should review the selections of /bin files.

4.3.3 Making the initrd (minimal root file system)

When you are done customizing these to files, su to root and run :

```
mk_initrd
```

This script creates the initial root file system that will be mounted in a RAM disk. It works in four phases. First it processes the util.conf and common.sh files setting up variables and defining common functions. Next, it creates a ramdisk, 10 MB in size, so you will propably have to set your ramdisk_size variable in lilo's 'append' string (like append="ramdisk_size=10240").

```
vermeer[~]# dd if=/dev/zero of=/dev/ram bs=1k count=10240
10240+0 records in
10240+0 records out
```

We have finally decided to use ext2 as its filesystem, since minix and romfs has too many restrictions. The RAM disk is formatted with mke2fs, using a 1024 inode ratio. In order to maintain maximum caompatibility, all device files are copied from the debian filesystem to /dev, hence this inode ratio.

```
vermeer[~]# mke2fs -i 1024 -vm1 /dev/ram 10240
mke2fs 1.17, 26-Oct-1999 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1024 inodes, 1024 blocks
10 blocks (0.98%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
1024 inodes per group

Writing inode tables: done

Writing superblocks and filesystem accounting information: done
```

As a next step, the formatted RAM disk is mounted and the directory and link structure is made:

```
vermeer[~]# mkdir -p $__mnt
vermeer[~]# mount -t ext2 /dev/ram $__mnt
...
```

Among the directories are /init_bin, /init_lib and /installroot. Note that the initrd is only a temporary means. The links /bin, /lib are pointing to /init_bin, /init_lib; after the installroot is mounted, they will be relinked to the new location (bin -> /installroot/bin, lib -> /installroot/lib). Note: the re-linking will take place in the linuxrc script. Other links, like /etc, /sbin, /usr, point inside the installroot directory right from the start (respectively installroot/etc, ..). As the client boots, it needs the files /etc/mtab and /etc/fstab, so we simply create them :

```
touch $__mnt/installroot/etc/{passwd,group,fstab}
```

After the installroot is mounted, all files and directories previously located in /installroot will become invisible as long as this file system remains attached, thus all created links will point at valid (albeit readonly) directories.

Next, the script copies binaries specified in $__binaries to /init_bin. It looks for required libraries as well as the adequate loaders and copies them to /init_lib. Note: the ldd command (checks the shared library dependencies of a binary) is executed in a chroot (debian) environment of the installroot, so it is independent of your local linux distribution:

```
chroot $INSTALL_ROOT ldd $bin | while read ...
```

Notes on BusyBox usage: When you create a link to BusyBox for the utility you wish to use, when BusyBox is called using that link it will behave as if the command itself has been invoked. For example, entering

```
ln -s ./BusyBox ls
./ls
```

will cause BusyBox to behave as 'ls' (if the 'ls' command has been compiled into BusyBox).

The BusyBox multicall-library has been compiled during the installroot setup. One of the resulting files - busybox.links - contains all links corresponding to the functions compiled into BusyBox.

The script copies the busybox executable to /init_bin and creates all required links :

```
sed 's~.*/~~' $INSTALL_ROOT/$__busy_links |
while read link
do
        ln -s busybox $link
done
```

where $__busy_links contains the location of busybox.links.

Finally, the neccessary scripts, in particular linuxrc, and dhclient configuration files are copied from the installroot.

Now that the initrd has been created, we unmount the ramdisk, 'dd' it to a file and finally, compress it :

```
umount $__mnt
dd if=/dev/ram bs=1k count=10240 | gzip -v9 > $__target
```

mk_initrd will produce occasional messages showing what it is doing. Any errors should be prominent.

When you are satisfied with the output from mk_initrd, create an appropriate bootdisk. Refer to section 5.1 (Booting from floppy) for further details.

4.3.4 Benediction

You are done. Try to boot your client using your favourite method (lilo, syslinux, pxelinux, netboot etc.). If all has gone well, the linuxrc script starts.

4.3.5 ...and what does the linuxrc script do anyway?

You should see :

```
starting NAIS (initrd.gz) ...
```

Linuxrc performs only a few tasks:

- the dhclient receives information from a dhcp server: it sets the NAIS_SERVER, NAIS_PATH, NAIS_ROOTPATH variables accordingly to the available network host and the appropriate directories (`I: mounting NAIS ...`).
- the remote /nais and /installroot directories are being mounted on their local counterparts. Then we relink the /bin and /lib directories (bin->init_bin, lib->init_lib) to the /installroot (`I: changing lib ...`).
- at this point, the actual install script starts ($INIT_SCRIPT).

Finally, when linuxrc and all subsequently launched (forked) scripts terminate, the client's newly made root file system is mounted and the initrd is moved to /initrd.

5 How the client boots

There are two methods for booting the client. The computer can boot from its network interface card (NIC) to receive the boot images via DHCP/TFTP or a suitable kernel as well as an initrd image is loaded from a floppy. Please refer to section 4 (Setting up the client's resources) for details on how an appropriate initial ramdisk image is built and what is required of a feasible client-kernel.

Throughout this section let `initrd.gz` denote a gzipped initrd image and `bzImage` a client-kernel, just like mentioned above.

5.1 Booting from floppy

For testing/rescue purposes or because your NIC does not have a PROM (yet) you can build a boot floppy to use with NAIS. We will only describe how *lilo(8)* and *syslinux* may be used to achieve this. For detailed information see *initrd.txt* <http://www.kernelnotes.org/doc22/initrd.txt>, *ramdisk.txt* <http://www.kernelnotes.org/doc22/ramdisk.txt>, *kernel-parameters.txt* <http://www.kernelnotes.org/doc22/kernel-parameters.txt>, and the *Bootdisk-HOWTO* <http://linuxdoc.org/HOWTO/Bootdisk-HOWTO/> as well as the documentation contained within both packages.

5.1.1 syslinux

This Linux boot loader operates off an MS-DOS ®/Windows® FAT filesystem. This makes it rather easy to maintain your boot floppies using standard MS-DOS ® tools. Its second feature, which is more important to us, is the usability for booting Linux of a network server. To create a boot floppy type:

```
vermeer[/tmp]# superformat /dev/fd0u1440
vermeer[/tmp]# mount /dev/fd0u1440 /floppy/
vermeer[/tmp]# cp bzImage initrd.gz /floppy/
vermeer[/tmp]# cat <<EOF > /floppy/syslinux.cfg
? DEFAULT bzImage
? TIMEOUT 0
? APPEND $options
? EOF
vermeer[/tmp]# umount /floppy/
vermeer[/tmp]# syslinux /dev/fd0u1440
```

Here `$options` means the kernel parameters that are passed to the kernel command line. In the above example we used `options="rw ramdisk_size=10240 initrd=initrd.gz reboot=warm vga=normal"` . NOTE: All kernel parameters are case sensitive and the value of `"ramdisk_size"` MUST correspond to the size of your initrd. See section 4.3 (How to create the Initial RAM disk) for details.

5.1.2 lilo

To build a lilo boot disk that contains a `bzImage` and an `initrd.gz` you have to supply a valid `lilo.conf`:

```
boot      = /dev/fd0u1440
install   = /boot/boot.b  # from your lilo-package
map       = /boot/map     # created when running /sbin/lilo
backup    = /dev/null
compact
read-write
vga       = normal
image     = bzImage
label     = NAIS
root      = /dev/fd0u1440
initrd    = initrd.gz
append    = "ramdisk=10240 reboot=warm"
```

Now you can build your boot floppy by typing:

```
vermeer[/tmp]# mke2fs -i 8192 -m 0 /dev/fd0u1440
vermeer[/tmp]# mount /dev/fd0u1440 /floppy/
vermeer[/tmp]# rm -rf /floppy/lost+found/
vermeer[/tmp]# mkdir /floppy/{dev,boot}
vermeer[/tmp]# cp -a /dev/{fd0u1440,null} /floppy/dev/
vermeer[/tmp]# cp /boot/boot.b /floppy/boot/
vermeer[/tmp]# cp initrd.gz bzImage lilo.conf /floppy/
vermeer[/tmp]# lilo -v -C lilo.conf -r /floppy/
```

5.2 Booting from network card

For administrative purposes, booting from network card (NIC) is much more suitable than booting from floppy. In order to use this boot method, the client's NIC needs a boot PROM that is able to communicate with a DHCP server to receive communication-related configuration parameters such as network addresses and which is capable of communicating with a TFTP-server to get a boot image. Furthermore, it must be

guaranteed that the transmitted boot image is executed properly in terms of what the boot PROM expects in a boot image.

Note: All described netbooting methods require that your DHCP server is properly set up. Please see section 3 (Setting up the server) for details.

5.2.1 etherboot and netboot

etherboot <http://etherboot.sourceforge.net/> and netboot <http://www.han.de/~gero/netboot.html> are capable of creating a PROM binary (which must still be programmed onto a PROM) and a corresponding "tagged" TFTP boot image which includes a `bzImage` (and an optional `initrd.gz`). Some tools exist that help test a boot PROM image, in fact the support utilities are pretty much common to both etherboot and netboot.

The advantage of netboot is its ability to emulate just enough of a `DOS` environment such that unmodified `DOS` packet driver binaries – these are usually provided with the NIC – can be used for building a boot PROM. Thus, netboot supports a wider range of NICs. etherboot, on the other hand, creates smaller boot PROM images; the compressed versions will fit in 8 KB size (which all NIC's should support). Also etherboot does autoprobing of the hardware addresses while netboot only does autoprobing as long as the packet driver supports this feature. However, if you choose to use either one of these tools you should definitly read the documentation or take a look at the mailinglist <http://www.han.de/~gero/netboot/archive/maillist.html> ; the *Diskless-HOWTO* <http://linuxdoc.org/HOWTO/Diskless-HOWTO.html> might definitely be helpful as well. Here we will only give an example to get the idea how it works.

netboot You probably do not need to compile netboot yourself, an `apt-get install netboot` or its equivalent for a rpm-based system should be sufficient. Once you have netboot running you have to get a working packet driver for the client's NIC. This MS-DOS ®-program usually comes with your NIC, look at netboot's homepage for some links. The packet driver must fit into 32KB or 64KB to be programmed onto a PROM. If the driver is too big try using programs such as `pkzip` to decrease the size of the executable.

If you have the packet driver you can build your PROM-image by typing `makerom`. You will be asked a couple of questions, including where netboot can find your packet driver (e.g. `3c90xpd.com` if you have a 3Com® 3c90b) and the command line arguments (e.g. `/I=96`) for it. That means under `DOS` you would simply type

```
A:\> 3c90xpd.com /I=96
```

to start the driver. This will create two files, `image.flo` and another file which is to be burned onto a PROM. Testing the boot PROM is done with `cat image.flo > /dev/fd0`; you can use this floppy to boot your netbootable client from.

The next task is to take the `bzImage` and `initrd.gz` and to turn them into a tagged image. Such an image has a special header that tells the network bootloader where the bytes go in memory and at what address to start the program. To **ma**ke a **n**etwork **b**ootable **i**mage type:

```
vermeer[/tmp]# mknbi-linux -x -i rom -k bzImage -r initrd.gz \
   -a "ramdisk_size=10240" -o /tftpboot/clientnbi
Kernel image file name  = bzImage
Output file name        = /tftpboot/clientnbi
Ramdisk image file name = initrd.gz
Kernel command line     = "auto rw root=/dev/nfs nfsroot=kernel \
   nfsaddrs=rom ramdisk_size=10240"
```

```
vermeer[/tmp]# chmod a=r,u+w /tftpboot/clientnbi
```

Now a tagged image `/tftpboot/clientnbi` with the printed parameters was built. Most used options of *mknbi-linux(8)* are obvious; `-x` means verbose and `-i rom` means all necessary ip addresses for NFS root mounting will be determined at runtime using the BOOTP answer the bootrom got from the DHCP server. Although NAIS does not use a NFS root this option might be useful for other purposes. Note that you can use vendor tag 129 (`option option-129`) to pass additional parameters to the kernel at runtime. The string value given with this tag is appended verbatim to the command line by the boot loader.

Test your PROM image i.e. boot your client from that floppy:

```
Disk loader for net boot
Uncopressing... done
.
.
Found packet driver at int 60
Free memory: 31
.
.
BOOTP: Sending request (press ESC to abort): .ok
Local IP: 134.95.10.160
Server IP: 134.95.10.140 (134.95.10.140)
File name: /tftpboot/clientnbi
.
.
Uncompressing Linux... Ok, booting the kernel.
```

etherboot The same works for etherboot like this:

```
vermeer[/tmp]# echo "Not done yet ..."
```

Testing the boot PROM works for etherboot pretty much the same (although *mknbi-linux* for etherboot is slightly different).

```
vermeer[/tmp]# echo "Please refer to the documentation meanwhile."
```

5.2.2 Using PXE boot-PROMs

Another option to etherboot and netboot is to use a PXE-compliant boot PROM. Different from the method mentioned above, when using Intel ®'s PXE specification `<http://developer.intel.com/ial/WfM/wfmspecs.htm>` you have to distinguish two things. One is to obtain a PXE boot PROM the other is two get a PXE remote-boot processor that is able to load a `bzImage` and an `initrd.gz`.

There is a free tool that is oriented towards building PXE boot PROMs, nilo `<http://nilo.sourceforge.net/>`. While this document is written, developments are made to the NILO project. Thus, we have not tested it yet, but give it a try!

Since many proprietary solutions are based upon PXE, you should not have any problems finding a suitable boot PROM. Take a look at `<http://etherboot.sourceforge.net/commercial.html>` for a list of some providers.

pxelinux pxelinux is a syslinux derivative which is still beta but works well for us. If you are already familiar with syslinux you will see that pxelinux operates in many ways like syslinux does. Please see section 3 (Setting up the server) for details on setting up your DHCP server for pxelinux.

First copy `pxelinux.bin` from the syslinux distribution to `/tftpboot/` on your TFTP server, as well as `bzImage` and `initrd.gz`. Keep in mind that these files must be world readable!

```
vermeer[~]# cp /usr/lib/syslinux/pxelinux.bin /tftpboot/
vermeer[~]# cp bzImage initrd.gz /tftpboot/
vermeer[~]# chmod a+r /tftpboot/*
```

Then, create the directory `/tftpboot/pxelinux.cfg/` with read/write permissions for your `${NAIS_USER}`:

```
vermeer[/tftpboot]# mkdir pxelinux.cfg
vermeer[/tftpboot]# chgrp linuxadm pxelinux.cfg/
vermeer[/tftpboot]# chmod a+r,g+w pxelinux.cfg/
```

The configuration files (the equivalent of `syslinux.cfg`) will reside in this directory. pxelinux, unlike BpBatch, does not use additional vendor tags to determine which config file to use. Instead, the configuration file name depends on the IP address of the booting machine. pxelinux will search for its config file on the TFTP server in the following way:

First, it will search for the config file using its own IP address in upper case hexadecimal, e.g. `134.95.10.160` → `/tftpboot/pxelinux.cfg/865F0AA0`. If that file is not found, it will remove one hex digit and try again. For `134.95.10.160`, if `865F0AA0` is not found, it will try `865F0AA`, `865F0A`, `865F0`, `865F`, `865`, `86`, and `8` in that order. Finally, it will try looking for a file named `default` (in lower case).

Note: This may not be as easily readable as host names, but using hexadecimal instead of decimal makes it easy to group alike clients together. Our 16 cluster clients have IP addresses from `134.95.10.160` to `134.95.10.175`. Now, when we want to install the cluster, we simply delete/remove `/tftpboot/pxelinux/865F0AA?` while `865F0AA` is a config file with defaults for installing a cluster client. When a client's installation is finished a file like e.g. `865F0AA0A` for `134.95.10.170` has to be created with this client's settings.

If you have problems setting up your client to work with pxelinux you should try to create a boot floppy with syslinux, like in section 5.1 (Booting from floppy). In most cases it should be sufficient to use exactly the same config file for pxelinux that you used for syslinux – in fact we do not know of any cases where it did not work. ;)

```
vermeer[~]# mount /floppy/
vermeer[~]# cp /floppy/syslinux.cfg /tftpboot/pxelinux.cfg/865F0AA
vermeer[~]# chmod a+r /tftpboot/pxelinux.cfg/865F0AA
vermeer[~]# mv /tftpboot/pxelinux.cfg/865F0AA? /tmp/
```

In this case we use the seven-digit string `865F0AA`, which to corresponds to hosts `134.95.10.160 - 175`, to install the whole cluster.

Booting pxelinux successfully should look like this:

```
CLIENT MAC ADDR: 00 10 5A 25 CB 73
CLIENT IP: 134.95.10.160  MASK: 255.255.255.0  DHCP IP: 134.95.10.140
GATEWAY IP: 134.95.10.254
```

```
PXELINUX 1.48 1999-09-26  Copyright (C) 1994-1999 H. Peter Anvin
PXE entry point found (we hope) at 9D98:00F6
My IP address seems to be 865F0AA0
TFTP prefix: /tftpboot/
Trying to load: pxelinux.cfg/865F0AA0
Trying to load: pxelinux.cfg/865F0AA
Loading bzImage...........
Loading initrd.gz.............

Ready to start kernel...
Uncompressing Linux... Ok, booting the kernel.
```

BpBatch BpBatch is a non-free remote-boot processor that is free for personal use. This tool can perform a large variety of actions on a computer at boot-time before any operating system operation has started.

You have to set some vendor tags to make BpBatch work. Please see section 3 (Setting up the server) for all details; BpBatch's webpage *BpBatch Forum* <http://www.bpbatch.org/forum/> is also helpful as well the *Remote-Boot mini-HOWTO* <http://cui.unige.ch/info/pc/remote-boot/howto.html> .

First copy `bpbatch.P` (we call it `bpbatch`), `bpbatch.ovl` and `bpbatch.hlp` from the *BpBatch distribution* <http://www.bpbatch.org/downloads/bpb-exe.tar.gz> to `/tftpboot/` on your TFTP server, as well as `bzImage` and `initrd.gz`. Please check at all times that the required files in `/tftpboot/` are world readable!

```
vermeer[/tmp/bpb]# tar xfz /files/install/tars/bpb-exe.tar.gz
vermeer[/tmp/bpb]# cp bpbatch.P /tftpboot/bpbatch
vermeer[/tmp/bpb]# cp bpbatch.ovl bpbatch.hlp /tftpboot/
vermeer[/tmp/bpb]# chmod a+r-w /tftpboot/bpbatch* /tftpboot/{bzImage,initrd.gz}
```

The role of the DHCP server is to give the client an IP address and to make it load the file named `bpbatch` from the TFTP server. Thus add an entry in the DHCP configuration file for your client, with the boot file set to `"bpbatch"`. Define a vendor option tag 135 (decimal) set to `client` (on the ISC DHCP server, this is done by `option option-135 "/tftpboot/client"`). This batch script is interpreted an run by bpbatch, you may change the value of `option-135` to `"-i"` for interactive mode. NOTE: BpBatch always appends the suffix `.bpb` to the basename of the value of `option-135` (e.g. if you pass `"/tftpboot/client.test"` as option, BpBatch will try to get `client.bpb`).

You have to provide two different config files for use with BpBatch. The first, e.g. `client-install.bpb`, will be used to install your client, the second, e.g. `client-run.bpb`, will be used for running the client.

A feasible `client-install.bpb` with `"options"="auto rw root=/dev/nfs ramdisk_size=10240 reboot=warm vga=normal"` looks like:

```
set cachenever="on"
LinuxBoot "bzImage" "options" "initrd.gz"
```

Please note that BpBatch's special variable called `"CacheNever"` should always be turned on since we do not want BpBatch to try to cache the kernel image on the client's hard disk.

Now booting your client should look like this:

```
CLIENT MAC ADDR: 00 10 5A 25 CB 73
CLIENT IP: 134.95.10.160  MASK: 255.255.255.0  DHCP IP: 134.95.10.140
```

```
GATEWAY IP: 134.95.10.254

Starting BpBatch - PXE Boot ROM detected
BpBatch overlay loader v1.1 (Feb 11 2000)
Overlay file successfully loaded
.
.
- BootProm detected, using 134.95.10.140 as standard TFTP server
- Advanced Power Management V1.2
- Using up to 368K of conventional memory for the heap
- Using up to 15296K of extended memory
- Direct disk write access enabled
Linux 2.2.15 (roott@vermeer) #6 SMP Fri May 12 11:19:05 CEST 2000
Loading linux ramdisk...................

Loading.....................................................
Uncompressing Linux... Ok, booting the kernel.
```

6 The installation process, 1st part

This section will be covered in a future release... until then, refer to other sections as well as scripts, sorry.

7 The installation process, 2nd part

This section will be covered in a future release... until then, refer to other sections as well as scripts, sorry.

8 How configuring works

This section will be covered in a future release... until then, refer to other sections as well as scripts, sorry.

9 Resources

- BusyBox <http://busybox.lineo.com/>. This utility combines tiny versions of many common UNIX utilities into a single small executable. They contain fewer options than their full featured GNU counterparts; however, the options that are included provide the expected functionality.
- Debian GNU/Linux base file system - base2_2.tgz <ftp://ftp.debian.org/debian/dists/potato/main/disks-i
  This contains a complete minimalist Debian GNU/Linux installation, as well as resources required to begin an installation of other needed utilities.
- Additional kernel modules and the modconf tool - drivers.tgz (same url as above)
- ash, Keith Almquist's tiny Bourne shell clone, used in the initrd. It is a version of sh with features similar to those of the System V shell.
- freeramdisk, a utility that comes with the loadlin package. Since the initrd-image is created within a RAM-disk device, this utility is needed to release this resource. Further information on loadlin can be obtained at <ftp://elserv.ffm.fgan.de/pub/linux/loadlin-1.6/> A very simple ex-

ample for building an image for initrd, also including the program 'freeramdisk', can be found on <ftp://elserv.ffm.fgan.de/pub/linux/loadlin-1.6/initrd-example.tgz>

- lilo, a boot loader, used to create floppy disks for testing purposes. Current versions are available at <ftp://lrcftp.epfl.ch/pub/people/almesber/lilo/>
- syslinux <ftp://ftp.kernel.org/pub/linux/utils/boot/syslinux/> , a linux boot loader, which can operate off an MS-DOS®/Windows® FAT filesystem. Configuration is largely equivalent to lilo's. Even more important: it is similar to pxelinux.
- BpBatch <http://www.bpbatch.org/> is a non-free remote-boot processor that is free for personal use. This tool can perform a large variety of actions on a computer at boot-time before any operating system operation has started.
- netboot <http://www.han.de/~gero/netboot.html> is capable of creating a PROM binary (which must still be programmed onto a PROM) and a corresponding "tagged" TFTP boot image which includes a bzImage (and an optional initrd.gz).
- etherboot <http://etherboot.sourceforge.net/> , a netboot alternative.
- cfengine <http://www.iu.hioslo.no/cfengine> , a high level language designed for testing and configuring unix-like systems attached to a TCP/IP network.
- perl - Practical Extraction and Report Language.
- TFTP server which supports DARPA Trivial File Transfer Protocol
- DHCP server and client, a boot protocol used to assign IP addresses, set up DNS nameservers. This utility was written by Ted Lemon *mellon@vix.com* under a contract with Vixie Labs. Funding for this project was provided by the Internet Software Consortium <http://www.isc.org/isc> .

10 Acknowledgements

The following people have contributed substantially to NAIS. If we missed somebody *please* send an email to *nais@informatik.uni-koeln.de* and we will fix it:

Not done yet ;)

11 Disclaimer and Copyright

We require that you properly attribute the authors of this document on any materials derived from this document. If you modify and improve this document, we request that you notify the authors of this document via *nais@informatik.uni-koeln.de*.

12 Glossary

Some commonly used abbreviations and their meanings:

ROM

ROM is "built-in" computer memory containing data that normally can only be read, not written to. ROM contains the programming that allows your computer to be "booted up" or regenerated each time you turn it on. Unlike a computer's random access memory (RAM), the data in ROM is not lost when the computer power is turned off. The ROM is sustained by a small long-life battery in your computer.

PROM

PROM (programmable read-only memory) is read-only memory (ROM) that can be modified once by a user. PROM is a way of allowing a user to tailor a microcode program using a special machine called a PROM programmer. This machine supplies an electrical current to specific cells in the ROM that effectively blows a fuse in them. The process is known as burning the PROM. Since this process leaves no margin for error, most ROM chips designed to be modified by users use erasable programmable read-only memory (EPROM) or electrically erasable programmable read-only memory (EEPROM).

EPROM

EPROM (erasable programmable read-only memory) is programmable read-only memory (PROM) that can be erased and re-used. Erasure is caused by shining an intense ultraviolet light through a window that is designed into the memory chip. (Although ordinary room lighting does not contain enough ultraviolet light to cause erasure, bright sunlight can cause erasure. For this reason, the window is usually covered with a label when not installed in the computer.)

EEPROM

EEPROM (electrically erasable programmable read-only memory) is user-modifiable read-only memory (ROM) that can be erased and reprogrammed (written to) repeatedly through the application of higher than normal electrical voltage. Unlike EPROM chips, EEPROMs do not need to be removed from the computer to be modified. However, an EEPROM chip has to be erased and reprogrammed in its entirety, not selectively. It also has a limited life - that is, the number of times it can be reprogrammed is limited to tens or hundreds of thousands of times. In an EEPROM that is frequently reprogrammed while the computer is in use, the life of the EEPROM can be an important design consideration.