Gentoo Linux amd64 Handbook: Network configuration

From Gentoo Wiki
Handbook:AMD64 (/wiki/Special:MyLanguage/Handbook:AMD64) | Full (/wiki/Special:MyLanguage/Handbook:AMD64/Full)

Contents

Getting started

Note
This document assumes that the user has correctly configured his kernel, the for his hardware and the user knows the interface name of the hardware. We also assume that the interface to configure is eth0, which is just as an example as it could also be eno0, ens1, wlan0, enp1s0 etc.

To get started configuring the network card, tell the Gentoo RC system about it. This is done by creating a symbolic link from `net.lo` to `net.eth0` (or whatever the network interface name is on the system) in `/etc/init.d`.

```
root # cd /etc/init.d
root # ln -s net.lo net.eth0
```

Gentoo's RC system now knows about that interface. It also needs to know how to configure the new interface. All the network interfaces are configured in `/etc/conf.d/net`. Below is a sample configuration for DHCP and static addresses.

FILE `/etc/conf.d/net` **Example network configuration**

```
# For DHCP
config_eth0="dhcp"

# For static IP using CIDR notation
config_eth0="192.168.0.7/24"
routes_eth0="default via 192.168.0.1"
dns_servers_eth0="192.168.0.1 8.8.8.8"

# For static IP using netmask notation
config_eth0="192.168.0.7 netmask 255.255.255.0"
routes_eth0="default via 192.168.0.1"
dns_servers_eth0="192.168.0.1 8.8.8.8"
```

Note
If no configuration is mentioned for an interface then DHCP is assumed.

Note
CIDR stands for Classless InterDomain Routing. Originally, IPv4 addresses were classified as A, B, or C. The early classification system did not envision the massive popularity of the Internet, and is in danger of running out of new unique addresses. CIDR is an addressing scheme that allows one IP address to designate many IP addresses. A CIDR IP address looks like a normal IP address except that it ends with a slash followed by a number; for example, 192.168.0.0/16. CIDR is described in RFC 1519 (//tools.ietf.org/html/rfc1519).

Now that the interface is configured, we can start and stop it using the following commands:

```
root # /etc/init.d/net.eth0 start
root # /etc/init.d/net.eth0 stop
```

Important
When troubleshooting networking, take a look at `/var/log/rc.log`. Unless rc_logger is set to "NO" in `/etc/rc.conf`, information on the boot activity will be stored in that log file.

Now that the network interface has been successfully stopped and started, the next step is to have it started when Gentoo boots. Here's how to do this.

```
root # rc-update add net.eth0 default
root # rc
```

Note
The last "rc" command instructs Gentoo to start any scripts in the current runlevel that have not yet been started.

Advanced configuration

The config_eth0 variable is the heart of an interface configuration. It's a high level instruction list for configuring the interface (eth0 in this case). Each command in the instruction list is performed sequentially. The interface is deemed OK if at least one command works.

Here's a list of built-in instructions.

Command	Description
null	Do nothing
noop	If the interface is up and there is an address then abort configuration successfully
an IPv4 or IPv6 address	Add the address to the interface
dhcp, adsl or apipa (or a custom command from a 3rd party module)	Run the module which provides the command. For example dhcp will run a module that provides DHCP which can be one of either dhcpcd, dhclient or pump.

If a command fails, specify a fallback command. The fallback has to match the config structure exactly.

It is possible to chain these commands together. Here are some real world examples.

FILE `/etc/conf.d/net` **Configuration examples**

```
# Adding three IPv4 addresses
config_eth0="192.168.0.2/24
192.168.0.3/24
192.168.0.4/24"

# Adding an IPv4 address and two IPv6 addresses
config_eth0="192.168.0.2/24
4321:0:1:2:3:4:567:89ab
4321:0:1:2:3:4:567:89ac"

# Keep our kernel assigned address, unless the interface goes
# down so assign another via DHCP. If DHCP fails then add a
# static address determined by APIPA
config_eth0="noop
dhcp"
fallback_eth0="null
apipa"
```

Note
When using the ifconfig module and adding more than one address, interface aliases are created for each extra address. So with the above two examples users will get interfaces eth0, eth0:1 and eth0:2. It is not possible to do anything special with these interfaces as the kernel and other programs will just treat eth0:1 and eth0:2 as eth0.

Important
The fallback order is important! If the null option was not specified then the apipa command would only be run if the noop command failed.

Note
APIPA and DHCP are discussed later.

Network dependencies

Init scripts in `/etc/init.d/` can depend on a specific network interface or just "net". All network interfaces in Gentoo's init system provide what is called "net".

If, in `/etc/rc.conf`, the rc_depend_strict variable is set to "YES", then all network interfaces that provide "net" *must* be active before a dependency on "net" is assumed to be met. In other words, if a system has a net.eth0 and net.eth1 and an init script depends on "net", then both must be enabled.

On the other hand, if rc_depend_strict="NO" is set, then the "net" dependency is marked as resolved the moment at least one network interface is brought up.

But what about net.br0 depending on net.eth0 and net.eth1? net.eth1 may be a wireless or PPP device that needs configuration before it can be added to the bridge. This cannot be done in `/etc/init.d/net.br0` as that's a symbolic link to net.lo.

The answer is to define an `rc_need_` setting in `/etc/conf.d/net`:

FILE `/etc/conf.d/net` **Adding a net.br0 dependency**

```
rc_need_br0="net.eth0 net.eth1"
```

That alone, however, is not sufficient. Gentoo's networking init scripts use a virtual dependency called "net" to inform the system when networking is available. Clearly, in the above case, networking should only be marked as available when net.br0 is up, not when the others are. So we need to tell that in `/etc/conf.d/net` as well:

FILE `/etc/conf.d/net` **Updating virtual dependencies and provisions for networking**

```
rc_net_lo_provide="!net"
rc_net_eth0_provide="!net"
rc_net_eth1_provide="!net"
```

For a more detailed discussion about dependency, consult the section on writing initscripts in the Gentoo Handbook. More information about `/etc/rc.conf` is available as comments within that file.

Variable names and values

Variable names are dynamic. They normally follow the structure of `variable_${interface|mac|essid|apmac}`. For example, the variable `dhcpcd_eth0` holds the value for dhcpcd options for eth0 and `dhcpcd_essid` holds the value for dhcpcd options when any interface connects to the ESSID "essid".

However, there is no hard and fast rule that states interface names must be ethx. In fact, many wireless interfaces have names like wlanx, rax as well as ethx. Also, some user defined interfaces such as bridges can be given any name. To make life more interesting, wireless Access Points can have names with non alpha-numeric characters in them - this is important because users can configure networking parameters per ESSID.

The downside of all this is that Gentoo uses bash variables for networking - and bash cannot use anything outside of English alpha-numerics. To get around this limitation we change every character that is not an English alpha-numeric into an _ (underscore) character.

Another downside of bash is the content of variables - some characters need to be escaped. This can be achieved by placing the \ character in front of the character that needs to be escaped. The following list of characters needs to be escaped in this way: `"`, `'` and `\`.

In this example we use wireless ESSID as they can contain the widest scope of characters. We shall use the ESSID *My "\ NET*:

FILE `/etc/conf.d/net` **Variable names**

```
# This does work, but the domain is invalid
dns_domain_My____NET="My \"\\ NET"
```

The above sets the DNS domain to *My "\ NET* when a wireless card connects to an AP whose ESSID is *My "\ NET*.

Network interface naming

How it works

Network interface names are not chosen arbitrarily: the Linux kernel and the device manager (most systems have udev as their device manager although others are available as well) choose the interface name through a fixed set of rules.

When an interface card is detected on a system, the Linux kernel gathers the necessary data about this card. This includes:

- the onboard (on the interface itself) registered name of the network card, which is later seen through the ID_NET_NAME_ONBOARD parameter;
- the slot in which the network card is plugged in, which is later seen through the ID_NET_NAME_SLOT parameter;
- the path through which the network card device can be accessed, which is later seen through the ID_NET_NAME_PATH parameter;

- the (vendor-provided) MAC address of the card, which is later seen through the ID_NET_NAME_MAC parameter;

Based on this information, the device manager decides how to name the interface on the system. By default, it uses the first hit of the first three parameters above (ID_NET_NAME_ONBOARD, _SLOT or _PATH). For instance, if ID_NET_NAME_ONBOARD is found and set to eno1, then the interface will be called eno1.

Given an active interface name, the values of the provided parameters can be shown using `udevadm` :

```
root # udevadm test-builtin net_id /sys/class/net/enp3s0 2>/dev/null
```

```
ID_NET_NAME_MAC=enxc80aa9429d76
ID_OUI_FROM_DATABASE=Quanta Computer Inc.
ID_NET_NAME_PATH=enp3s0
```

As the first (and actually only) hit of the top three parameters is the ID_NET_NAME_PATH one, its value is used as the interface name. If none of the parameters is found, then the system reverts back to the kernel-provided naming (eth0, eth1, etc.)

Using the old-style kernel naming

Before this change, network interface cards were named by the Linux kernel itself, depending on the order that drivers are loaded (amongst other, possibly more obscure reasons). This behavior can still be enabled by setting the net.ifnames=0 boot option in the boot loader.

Using custom names

The entire idea behind the change in naming is not to confuse people, but to make changing the names easier. Suppose a system has two interfaces that are otherwise called eth0 and eth1. One is meant to access the network through a wire, the other one is for wireless access. With the support for interface naming, users can have these called lan0 (wired) and wifi0 (wireless - it is best to avoid using the previously well-known names like eth* and wlan* as those can still collide with the suggested names).

Find out what the parameters are for the cards and then use this information to set up a custom own naming rule:

```
root # udevadm test-builtin net_id /sys/class/net/eth0 2>/dev/null
```

```
ID_NET_NAME_MAC=enxc80aa9429d76
ID_OUI_FROM_DATABASE=Quanta Computer Inc.
```

```
root # vim /etc/udev/rules.d/70-net-name-use-custom.rules
```

```
# First one uses MAC information, and 70- number to be before other net rules
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="c8:0a:a9:42:9d:76", NAME="lan0"
```

```
root # vim /etc/udev/rules.d/76-net-name-use-custom.rules
```

```
# Second one uses ID_NET_NAME_PATH information, and 76- number to be between
# 75-net-*.rules and 80-net-*.rules
SUBSYSTEM=="net", ACTION=="add", ENV{ID_NET_NAME_PATH}=="enp3s0", NAME="wifi0"
```

Because the rules are triggered before the default one (rules are triggered in alphanumerical order, so 70 comes before 80) the names provided in the rule file will be used instead of the default ones. The number granted to the file should be between 76 and 79 (the environment variables are defined by a rule start starts with 75 and the fallback naming is done in a rule numbered 80).

Network modules

We now support modular networking scripts, which means we can easily add support for new interface types and configuration modules while keeping compatibility with existing ones.

Modules load by default if the package they need is installed. If users specify a module here that doesn't have its package installed then they get an error stating which package they need to install. Ideally, the modules setting is only used when two or more packages are installed that supply the same service and one needs to be preferred over the other.

> **Note**
> All settings discussed here are stored in `/etc/conf.d/net` unless otherwise specified.

FILE `/etc/conf.d/net` **Module definitions**

```
# Prefer ifconfig over iproute2
modules="ifconfig"

# You can also specify other modules for an interface
# In this case we prefer pump over dhcpcd
modules_eth0="pump"

# You can also specify which modules not to use - for example you may be
# using a supplicant or linux-wlan-ng to control wireless configuration but
# you still want to configure network settings per ESSID associated with.
modules="!iwconfig"
```

Interface handlers

We provide two interface handlers presently: ifconfig and iproute2. Only one of these is needed to do any kind of network configuration.

ifconfig is installed by default (the net-tools package is part of the system profile). iproute2 is a more powerful and flexible package, but it's not included by default.

```
root # emerge --ask sys-apps/iproute2
```

FILE `/etc/conf.d/net` **iproute2 is installed but still prefer ifconfig**

```
# To prefer ifconfig over iproute2 if both are installed as openrc prefers
# to use iproute2 then
modules="ifconfig"
```

As both ifconfig and iproute2 do very similar things we allow their basic configuration to work with each other. For example both the below code snippet work regardless of which module the user is using.

FILE `/etc/conf.d/net` **Example different approaches for configuration**

```
config_eth0="192.168.0.2/24"
config_eth0="192.168.0.2 netmask 255.255.255.0"

# We can also specify broadcast
config_eth0="192.168.0.2/24 brd 192.168.0.255"
config_eth0="192.168.0.2 netmask 255.255.255.0 broadcast 192.168.0.255"
```

DHCP

DHCP is a means of obtaining network information (IP address, DNS servers, Gateway, etc) from a DHCP server. This means that if there is a DHCP server running on the network, the user just has to tell each client to use DHCP and it sets up the network all by itself. Of course, the user will have to configure for other things like wireless, PPP or other things if required before he can use DHCP.

DHCP can be provided by dhclient, dhcpcd, or pump. Each DHCP module has its pros and cons - here's a quick run down.

DHCP Module	Package	Pros	Cons
dhclient	net-misc/dhcp (http://packages.gentoo.org/package/net-misc/dhcp)	Made by ISC, the same people who make the BIND DNS software. Very configurable	Configuration is overly complex, software is quite bloated, cannot get NTP servers from DHCP, does not send hostname by default
dhcpcd	net-misc/dhcpcd (http://packages.gentoo.org/package/net-misc/dhcpcd)	Long time Gentoo default, no reliance on outside tools, actively developed by Gentoo	Can be slow at times, does not yet daemonize when lease is infinite
pump	net-misc/pump (http://packages.gentoo.org/package/net-misc/pump)	Lightweight, no reliance on outside tools	No longer maintained upstream, unreliable, especially over modems, cannot get NIS servers from DHCP

If more than one DHCP client is installed, specify which one to use - otherwise we default to dhcpcd if available.

To send specific options to the DHCP module, use module_eth0="..." (change module to the DHCP module being used - i.e. dhcpcd_eth0).

We try and make DHCP relatively agnostic - as such we support the following commands using the dhcp_eth0 variable. The default is not to set any of them:

- release - releases the IP address for re-use
- nodns - don't overwrite `/etc/resolv.conf`
- nontp - don't overwrite `/etc/ntp.conf`
- nonis - don't overwrite `/etc/yp.conf`

FILE `/etc/conf.d/net` **Sample DHCP configuration**

```
# Only needed if you have more than one DHCP module installed
modules="dhcpcd"

config_eth0="dhcp"
dhcpcd_eth0="-t 10" # Timeout after 10 seconds
dhcp_eth0="release nodns nontp nonis" # Only get an address
```

Note
dhcpcd and pump send the current hostname to the DHCP server by default so this does not need to be specified anymore.

ADSL with PPPoE/PPPoA

First we need to install the ADSL software.

```
root # emerge --ask net-dialup/ppp
```

Second, create the PPP net script and the net script for the Ethernet interface to be used by PPP:

```
root # ln -s /etc/init.d/net.lo /etc/init.d/net.ppp0
root # ln -s /etc/init.d/net.lo /etc/init.d/net.eth0
```

Be sure to set rc_depend_strict to "YES" in `/etc/rc.conf`.

Now we need to configure `/etc/conf.d/net`.

FILE `/etc/conf.d/net` **A basic PPPoE setup**

```
config_eth0=null (Specify the ethernet interface)
config_ppp0="ppp"
link_ppp0="eth0" (Specify the ethernet interface)
plugins_ppp0="pppoe"
username_ppp0='user'
password_ppp0='password'
pppd_ppp0="
noauth
defaultroute
usepeerdns
holdoff 3
child-timeout 60
lcp-echo-interval 15
lcp-echo-failure 3
noaccomp noccp nobsdcomp nodeflate nopcomp novj novjccomp"

rc_need_ppp0="net.eth0"
```

It is also possible to set the password in `/etc/ppp/pap-secrets`.

FILE `/etc/ppp/pap-secrets` **Sample pap-secrets**

```
# The * is important
"username"  *  "password"
```

If PPPoE is used with a USB modem then make sure to emerge br2684ctl. Please read `/usr/portage/net-dialup/speedtouch-usb/files/README` for information on how to properly configure it.

Important
Please carefully read the section on ADSL and PPP in `/usr/share/doc/netifrc-*/net.example.bz2`. It contains many more detailed explanations of all the settings any particular PPP setup will likely need.

APIPA (Automatic Private IP Addressing)

APIPA tries to find a free address in the range 169.254.0.0-169.254.255.255 by arping a random address in that range on the interface. If no reply is found then we assign that address to the interface.

This is only useful for LANs where there is no DHCP server and the system doesn't connect directly to the Internet and all other computers use APIPA.

For APIPA support, emerge net-misc/iputils (http://packages.gentoo.org/package/net-misc/iputils) or net-analyzer/arping (http://packages.gentoo.org/package/net-analyzer/arping).

FILE `/etc/conf.d/net` **APIPA configuration**

```
# Try DHCP first - if that fails then fallback to APIPA
config_eth0="dhcp"
fallback_eth0="apipa"

# Just use APIPA
config_eth0="apipa"
```

Bonding

For link bonding/trunking emerge net-misc/ifenslave (http://packages.gentoo.org/package/net-misc/ifenslave).

Bonding is used to increase network bandwidth or to improve resiliency in face of hardware failures. If a system has two network cards going to the same network, then the administrator can bond them together so the applications see just one interface but they really use both network cards.

There are many ways to configure bonding. Some of them, such as the 802.3ad LACP mode, require support and additional configuration of the network switch. For a reference of the individual options, please refer to the local copy of `/usr/src/linux/Documentation/networking/bonding.txt`.

First, clear the configuration of the participating interfaces:

FILE `/etc/conf.d/net` **Clearing interface configuration**

```
config_eth0="null"
config_eth1="null"
config_eth2="null"
```

Next, define the bonding between the interfaces:

FILE `/etc/conf.d/net` **Define the bonding**

```
slaves_bond0="eth0 eth1 eth2"
config_bond0="192.168.100.4/24"
# Pick a correct mode and additional configuration options which suit your needs
mode_bond0="balance-alb"
```

Remove the net.eth* services from the runlevels, create a net.bond0 one and add that one to the correct runlevel.

Bridging (802.1d support)

For bridging support emerge net-misc/bridge-utils (http://packages.gentoo.org/package/net-misc/bridge-utils).

Bridging is used to join networks together. For example, a system may have a server that connects to the Internet via an ADSL modem and a wireless access card to enable other computers to connect to the Internet via the ADSL modem. It is possible to create a bridge to join the two interfaces together.

FILE `/etc/conf.d/net` **Bridge configuration**

```
# Configure the bridge - "man brctl" for more details
brctl_br0="setfd 0
sethello 2
stp on"

# To add ports to bridge br0
bridge_br0="eth0 eth1"

# You need to configure the ports to null values so dhcp does not get started
config_eth0="null"
config_eth1="null"

# Finally give the bridge an address - you could use DHCP as well
config_br0="192.168.0.1/24"

# Depend on eth0 and eth1 as they may require extra configuration
rc_need_br0="net.eth0 net.eth1"
```

Important
For using some bridge setups, consult the variable name documentation.

MAC address

It is possible to change the MAC address of the interfaces through the network configuration file too.

FILE `/etc/conf.d/net` **MAC Address change example**

```
# To set the MAC address of the interface
mac_eth0="00:11:22:33:44:55"

# To randomize the last 3 bytes only
mac_eth0="random-ending"

# To randomize between the same physical type of connection (e.g. fibre,
# copper, wireless) , all vendors
mac_eth0="random-samekind"

# To randomize between any physical type of connection (e.g. fibre, copper,
# wireless) , all vendors
mac_eth0="random-anykind"

# Full randomization - WARNING: some MAC addresses generated by this may
# NOT act as expected
mac_eth0="random-full"
```

Tunneling

Tunneling does not require any additional software to be installed as the interface handler can do it.

FILE `/etc/conf.d/net` **Tunneling configuration**

```
# For GRE tunnels
iptunnel_vpn0="mode gre remote 207.170.82.1 key 0xffffffff ttl 255"

# For IPIP tunnels
iptunnel_vpn0="mode ipip remote 207.170.82.2 ttl 255"

# To configure the interface
config_vpn0="192.168.0.2 peer 192.168.1.1"
```

VLAN (802.1q support)

For VLAN support, make sure that sys-apps/iproute2 (http://packages.gentoo.org/package/sys-apps/iproute2) is installed and ensure that iproute2 is used as configuration module rather than ifconfig.

Virtual LAN is a group of network devices that behave as if they were connected to a single network segment - even though they may not be. VLAN members can only see members of the same VLAN even though they may share the same physical network.

To configure VLANs, first specify the VLAN numbers in `/etc/conf.d/net` like so:

FILE `/etc/conf.d/net` **Specifying VLAN numbers**

```
vlans_eth0="1 2"
```

Next, configure the interface for each VLAN:

FILE `/etc/conf.d/net` **Interface configuration for each VLAN**

```
config_eth0_1="172.16.3.1 netmask 255.255.254.0"
routes_eth0_1="default via 172.16.3.254"

config_eth0_2="172.16.2.1 netmask 255.255.254.0"
routes_eth0_2="default via 172.16.2.254"
```

VLAN-specific configurations are handled by vconfig like so:

FILE `/etc/conf.d/net` **Configuring the VLANs**

```
vlan1_name="vlan1"
vlan1_ingress="2:6 3:5"
eth0_vlan1_egress="1:2"
```

Important
For using some VLAN setups, consult the variable name documentation.

Introduction

Wireless networking on Linux is usually pretty straightforward. There are two ways of configuring wifi: graphical clients, or the command line.

The easiest way is to use a graphical client once a desktop environment is installed. Most graphical clients, such as wicd and NetworkManager, are pretty self-explanatory. They offer a handy point-and-click interface that gets users on a network in just a few seconds.

Note
wicd offers a command line utility in addition to the main graphical interface. Install the net-misc/wicd (http://packages.gentoo.org/package/net-misc/wicd) package with the ncurses USE flag set. This `wicd-curses` utility is particularly useful for folks who don't use a gtk-based desktop environment, but still want an easy command line tool that doesn't require hand-editing configuration files.

Wireless can also be configured from the command line by editing a few configuration files. This takes a bit more time to setup, but it also requires the fewest packages to download and install. Since the graphical clients are mostly self-explanatory (with helpful screen shots at their home pages), we'll focus on the command line alternatives.

There are two tools that support command-line driven wireless configurations: net-wireless/wireless-tools (http://packages.gentoo.org/package/net-wireless/wireless-tools) and net-wireless/wpa_supplicant (http://packages.gentoo.org/package/net-wireless/wpa_supplicant). Of these two, net-wireless/wpa_supplicant (http://packages.gentoo.org/package/net-wireless/wpa_supplicant) is the preferred one. The important thing to remember is that wireless networks are configured on a global basis and not an interface basis.

The net-wireless/wireless-tools (http://packages.gentoo.org/package/net-wireless/wireless-tools) software supports nearly all cards and drivers, but it cannot connect to WPA-only Access Points. If the networks only offer WEP encryption or are completely open, then net-wireless/wireless-tools (http://packages.gentoo.org/package/net-wireless/wireless-tools) beats the other package over simplicity.

Warning
The linux-wlan-ng driver is not supported by baselayout at this time. This is because linux-wlan-ng has its own setup and configuration which is completely different to everyone else's. The linux-wlan-ng developers are rumored to be changing their setup over to wireless-tools, so when this happens then linux-wlan-ng can be used with baselayout.

Some wireless cards are deactivated by default. To activate them, please consult the hardware documentation. Some of these cards can be unblocked using the rfkill application. If that is the case, use `rfkill list` to see the available cards and `rfkill unblock INDEX` to activate the wireless functionality. If not, then the wireless card might need to be unlocked through a button, switch or special key combination on the laptop.

WPA supplicant

The WPA supplicant project (http://hostap.epitest.fi/wpa_supplicant) provides a package that allows users to connect to WPA enabled access points.

```
root # emerge --ask net-wireless/wpa_supplicant
```

Important
It is necessary to have CONFIG_PACKET enabled in the kernel for wpa_supplicant to work. To see if it is enabled on the current kernel, try:

```
root # zgrep CONFIG_PACKET /proc/config.gz
root # grep CONFIG_PACKET /usr/src/linux/.config
```

Note
Depending on the USE flags, wpa_supplicant can install a graphical interface written in Qt4, which will

integrate nicely with KDE. To get it, enable USE=qt4 for the net-wireless/wpa_supplicant (http://packages.gentoo.org/package/net-wireless/wpa_supplicant) package.

Next, configure `/etc/conf.d/net` so that the wpa_supplicant module is preferred over wireless-tools (if both are installed, wireless-tools is the default).

FILE `/etc/conf.d/net` **Force the use of wpa_supplicant**

```
# Prefer wpa_supplicant over wireless-tools
modules="wpa_supplicant"

# It's important to tell wpa_supplicant which driver it should
# be using as it's not very good at guessing yet
wpa_supplicant_eth0="-Dnl80211"
```

Note
When using the host-ap driver it is necessary to put the card in *Managed mode* before it can be used with wpa_supplicant correctly. This can be achieved by setting iwconfig_eth0="mode managed" in `/etc/conf.d/net`.

Next configure wpa_supplicant itself (which is a bit more tricky depending on how secure the Access Points are). The below example is taken and simplified from `/usr/share/doc/wpa_supplicant-<version>/wpa_supplicant.conf.gz` which ships with wpa_supplicant.

FILE `/etc/wpa_supplicant/wpa_supplicant.conf` **Somewhat simplified example**

```
# The below line not be changed otherwise wpa_supplicant refuses to work
ctrl_interface=/var/run/wpa_supplicant

# Ensure that only root can read the WPA configuration
ctrl_interface_group=0

# Let wpa_supplicant take care of scanning and AP selection
ap_scan=1

# Simple case: WPA-PSK, PSK as an ASCII passphrase, allow all valid ciphers
network={
  ssid="simple"
  psk="very secret passphrase"
  # The higher the priority the sooner we are matched
  priority=5
}

# Same as previous, but request SSID-specific scanning (for APs that reject
# broadcast SSID)
network={
  ssid="second ssid"
  scan_ssid=1
  psk="very secret passphrase"
  priority=2
}

# Only WPA-PSK is used. Any valid cipher combination is accepted
network={
```

```
  ssid="example"
  proto=WPA
  key_mgmt=WPA-PSK
  pairwise=CCMP TKIP
  group=CCMP TKIP WEP104 WEP40
  psk=06b4be19da289f475aa46a33cb793029d4ab3db7a23ee92382eb0106c72ac7bb
  priority=2
}

# Plaintext connection (no WPA, no IEEE 802.1X)
network={
  ssid="plaintext-test"
  key_mgmt=NONE
}

# Shared WEP key connection (no WPA, no IEEE 802.1X)
network={
  ssid="static-wep-test"
  key_mgmt=NONE
  # Keys in quotes are ASCII keys
  wep_key0="abcde"
  # Keys specified without quotes are hex keys
  wep_key1=0102030405
  wep_key2="1234567890123"
  wep_tx_keyidx=0
  priority=5
}

# Shared WEP key connection (no WPA, no IEEE 802.1X) using Shared Key
# IEEE 802.11 authentication
network={
  ssid="static-wep-test2"
  key_mgmt=NONE
  wep_key0="abcde"
  wep_key1=0102030405
  wep_key2="1234567890123"
  wep_tx_keyidx=0
  priority=5
  auth_alg=SHARED
}

# IBSS/ad-hoc network with WPA-None/TKIP
network={
  ssid="test adhoc"
  mode=1
  proto=WPA
  key_mgmt=WPA-NONE
  pairwise=NONE
  group=TKIP
  psk="secret passphrase"
}
```

Wireless tools

Initial setup and managed mode

The wireless tools project (http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html) provides a generic way to configure basic wireless interfaces up to the WEP security level. While WEP is a weak security method it's still prevalent in the world.

Wireless tools configuration is controlled by a few main variables. The sample configuration file below should describe all that is needed. One thing to bear in mind is that no configuration means "connect to the strongest unencrypted Access Point" - wireless tools will always try and connect the system to something.

```
root # emerge --ask net-wireless/wireless-tools
```

Note
Although users can store their wireless settings in `/etc/conf.d/wireless` this guide recommends to store them in `/etc/conf.d/net`.

Important
You will need to consult the variable name documentation.

FILE `/etc/conf.d/net` **Sample iwconfig setup**

```
# Prefer iwconfig over wpa_supplicant
modules="iwconfig"

# Configure WEP keys for Access Points called ESSID1 and ESSID2
# You may configure up to 4 WEP keys, but only 1 can be active at
# any time so we supply a default index of [1] to set key [1] and then
# again afterwards to change the active key to [1]
# We do this incase you define other ESSID's to use WEP keys other than 1
#
# Prefixing the key with s: means it's an ASCII key, otherwise a HEX key
#
# enc open specified open security (most secure)
# enc restricted specified restricted security (least secure)
key_ESSID1="[1] s:yourkeyhere key [1] enc open"
key_ESSID2="[1] aaaa-bbbb-cccc-dd key [1] enc restricted"

# The below only work when we scan for available Access Points

# Sometimes more than one Access Point is visible so we need to
# define a preferred order to connect in
preferred_aps="'ESSID1' 'ESSID2'"
```

Fine-tune AP selection

It is possible to add some extra options to fine-tune the AP selection, but these are not required.

One way is to configure the system so it only connects to preferred APs. By default if everything configured has failed and wireless-tools can connect to an unencrypted Access Point then it will. This can be controlled by the associate_order variable. Here's a table of values and how they control this.

Value	Description
any	Default behaviour
preferredonly	Only connect to visible APs in the preferred list
forcepreferred	Forceably connect to APs in the preferred order if they are not found in a scan
forcepreferredonly	Do not scan for APs - instead just try to connect to each one in order
forceany	Same as forcepreferred + connect to any other available AP

There is also the blacklist_aps and unique_ap selection. blacklist_aps works in a similar way to preferred_aps. unique_ap is a yes or no value that says if a second wireless interface can connect to the same Access Point as the first interface.

FILE `/etc/conf.d/net` **blacklist_aps and unique_ap example**

```
# Sometimes you never want to connect to certain access points
blacklist_aps="'ESSID3' 'ESSID4'"

# If you have more than one wireless card, you can say if you want
# to allow each card to associate with the same Access Point or not
# Values are "yes" and "no"
# Default is "yes"
unique_ap="yes"
```

Ad-hoc and master modes

To set the system up as an ad-hoc node when it fails to connect to any Access Point in managed mode, use this as a fallback:

FILE `/etc/conf.d/net` **Fallback to ad-hoc mode**

```
adhoc_essid_eth0="This Adhoc Node"
```

It is also possible to connect to ad-hoc networks, or to run the system in *master* mode so it becomes an access point itself.

FILE `/etc/conf.d/net` **Sample ad-hoc/master configuration**

```
# Set the mode - can be managed (default), ad-hoc or master
# Not all drivers support all modes
mode_eth0="ad-hoc"

# Set the ESSID of the interface
# In managed mode, this forces the interface to try and connect to the
# specified ESSID and nothing else
essid_eth0="This Adhoc Node"

# We use channel 3 if you don't specify one
channel_eth0="9"
```

Important
An important resource about channel selection is the BSD wavelan documentation (http://www.netbsd.org/Documentation/network/wavelan.html) found at the NetBSD documentation. There are 14 channels possible; We are told that channels 1-11 are legal for North America, channels 1-13 for most of Europe, channels 10-13 for France, and only channel 14 for Japan. If in doubt, please refer to the

documentation that came with the card or access point. Make sure that the channel selected is the same channel the access point (or the other card in an ad-hoc network) is on. The default for cards sold in North America and most of Europe is 3; the default for cards sold in France is 11, and the default for cards sold in Japan is 14.

Troubleshooting wireless tools

There are some more variables that can help to get the wireless up and running due to driver or environment problems. Here's a table of other things that can be tried.

Variable	Default Value	Description
iwconfig_eth0		See the iwconfig man page for details on what to send iwconfig
iwpriv_eth0		See the iwpriv man page for details on what to send iwpriv
sleep_scan_eth0	0	The number of seconds to sleep before attempting to scan. This is needed when the driver/firmware needs more time to active before it can be used.
sleep_associate_eth0	5	The number of seconds to wait for the interface to associate with the Access Point before moving onto the next one
associate_test_eth0	MAC	Some drivers do not reset the MAC address associated with an invalid one when they lose or attempt association. Some drivers do not reset the quality level when they lose or attempt association. Valid settings are MAC, quality and all.
scan_mode_eth0		Some drivers have to scan in ad-hoc mode, so if scanning fails try setting ad-hoc here
iwpriv_scan_pre_eth0		Sends some iwpriv commands to the interface before scanning. See the iwpriv man page for more details.
iwpriv_scan_post_eth0		Sends some iwpriv commands to the interface after scanning. See the iwpriv man page for more details.

Defining network configuration per ESSID

In this section, we show how to configure network settings based on the ESSID. For instance, with the wireless network with ESSID *ESSID1* configure a static IP address while ESSID *ESSID2* uses DHCP.

Note
This works with both wpa_supplicant as well as wireless-tools

Important
Please consult the variable name documentation.

FILE `/etc/conf.d/net` **override network settings per ESSID**

```
config_ESSID1="192.168.0.3/24 brd 192.168.0.255"
routes_ESSID1="default via 192.168.0.1"

config_ESSID2="dhcp"
fallback_ESSID2="192.168.3.4/24"
fallback_route_ESSID2="default via 192.168.3.1"

# We can define nameservers and other things too
# NOTE: DHCP will override these unless it's told not to
dns_servers_ESSID1="192.168.0.1 192.168.0.2"
dns_domain_ESSID1="some.domain"
dns_search_domains_ESSID1="search.this.domain search.that.domain"

# You override by the MAC address of the Access Point
# This handy if you goto different locations that have the same ESSID
config_001122334455="dhcp"
dhcpcd_001122334455="-t 10"
dns_servers_001122334455="192.168.0.1 192.168.0.2"
```

Standard function hooks

Four functions can be defined in `/etc/conf.d/net` which will be called surrounding the start/stop operations. The functions are called with the interface name first so that one function can control multiple adapters.

The return values for the preup() and predown() functions should be 0 (success) to indicate that configuration or de-configuration of the interface can continue. If preup() returns a non-zero value, then interface configuration will be aborted. If predown() returns a non-zero value, then the interface will not be allowed to continue de-configuration.

The return values for the postup() and postdown() functions are ignored since there's nothing to do if they indicate failure.

${IFACE} is set to the interface being brought up/down. ${IFVAR} is ${IFACE} converted to variable name bash allows.

FILE `/etc/conf.d/net` **pre/post up/down function examples**

```
 grep -q 'Link detected: no'; then
    ewarn "No link on ${IFACE}, aborting configuration"
    return 1
  fi

  # Remember to return 0 on success
  return 0
}

predown() {
  # The default in the script is to test for NFS root and disallow
  # downing interfaces in that case.  Note that if you specify a
  # predown() function you will override that logic.  Here it is, in
  # case you still want it...
  if is_net_fs /; then
    eerror "root filesystem is network mounted -- can't stop ${IFACE}"
    return 1
  fi

  # Remember to return 0 on success
  return 0
}

postup() {
  # This function could be used, for example, to register with a
  # dynamic DNS service.  Another possibility would be to
  # send/receive mail once the interface is brought up.
      return 0
}

postdown() {
  # This function is mostly here for completeness... I haven't
  # thought of anything nifty to do with it yet ;-)
  return 0
}
```

Note
For more information on writing functions, please read `/usr/share/doc/netifrc-*/net.example.bz2`.

Wireless tools function hook

Note
This will not work with WPA Supplicant - but the ${ESSID} and ${ESSIDVAR} variables are available in the postup() function.

Two functions can be defined in `/etc/conf.d/net` which will be called surrounding the associate function. The functions are called with the interface name first so that one function can control multiple adapters.

The return values for the preassociate() function should be 0 (success) to indicate that configuration or de-configuration of the interface can continue. If preassociate() returns a non-zero value, then interface configuration will be aborted.

The return value for the postassociate() function is ignored since there's nothing to do if it indicates failure.

${ESSID} is set to the exact ESSID of the AP the system is connecting to. ${ESSIDVAR} is ${ESSID} converted to a variable name bash allows.

FILE `/etc/conf.d/net` **pre/post association functions**

```
 grep -q 'Login incorrect'; then
      ewarn "Login Failed for ${user}"
      return 1
    fi
  fi

  return 0
}

postassociate() {
  # This function is mostly here for completeness... I haven't
  # thought of anything nifty to do with it yet ;-)

  return 0
}
```

Note
${ESSID} and ${ESSIDVAR} are unavailable in predown() and postdown() functions.

Note
For more information on writing custom functions, please read `/usr/share/doc/netifrc-*/net.example.bz2`.

Network management

With laptops, systems can be always on the move. As a result, the system may not always have an Ethernet cable or plugged in or an access point available. Also, the user may want networking to automatically work when an Ethernet cable is plugged in or an access point is found.

In this chapter, we cover how this can be done.

Note
This document only talks about ifplugd, but there are alternatives such as netplug. netplug is a lightweight alternative to ifplugd, but it relies on the kernel network drivers working correctly, and many drivers do not.

ifplugd

ifplugd (http://0pointer.de/lennart/projects/ifplugd/) is a daemon that starts and stops interfaces when an Ethernet cable is inserted or removed. It can also manage detecting association to Access Points or when new ones come in range.

```
root # emerge --ask sys-apps/ifplugd
```

Configuration for ifplugd is fairly straightforward too. The configuration is held in `/etc/conf.d/net`. Run `man ifplugd` for details on the available variables. Also, see `/usr/share/doc/netifrc-*/net.example.bz2` for more examples.

FILE `/etc/conf.d/net` **Sample ifplug configuration**

```
# Replace eth0 with the interface to be monitored
ifplugd_eth0="..."

# To monitor a wireless interface
ifplugd_eth0="--api-mode=wlan"
```

In addition to managing multiple network connections, users may want to add a tool that makes it easy to work with multiple DNS servers and configurations. This is very handy when the system receives its IP address via DHCP.

```
root # emerge --ask net-dns/openresolv
```

See `man resolvconf` to learn more about its features.