Configuring your network in Slackware

This article is intended as a reference guide to network card configuration in Slackware.
The network scripts themselves are well-documented (inside the scripts) but there is not much other written end-user documentation about what you put into the configuration files. The Network Configuration chapter in the *Slackware Linux Essentials* book explains in generic terms how Slackware's network configuration works, and how the use of DHCP (dynamic IP address assignment) differs from static IP's. I will try not to repeat what is written there.
There is another nice and freely available book on Slackware, called *Slackware Linux Basics*. This book should be considered as required follow-up reading material once you mastered the Slackware Essentials. The networking chapter is well worth reading.

In essence, my Wiki article documents the `/etc/rc.d/rc.inet1.conf` file. The only available documentation about the configurable network parameters used to be at the bottom of that file, and it took the shape of commented-out examples. In Slackware 12.2 two man pages were added, for rc.inet1 and rc.inet1.conf, both of which are based on this Wiki article.
The `rc.inet1` script in Slackware configures all your network interfaces - including wireless interfaces. If the `rc.inet1` script detects that it deals with a *wireless interface*, it will call the sub-script `rc.wireless` to configure this interface's wireless properties. Both scripts take their configuration information from the same file `rc.inet1.conf`.
I wrote a separate chapter about Wireless Networks because a wireless network interface has so many more configurable parameters than a “wired” interface. The configuration of WPA encryption a for wireless interface is documented in it's own chapter; the WPA parameters are taken from the file `/etc/wpa_supplicant.conf` instead of the `rc.inet1.conf` file.
The final section of this article looks at alternative network configuration managers (who usually come with a GUI based client program) that have become available for Slackware over time.

I will also try to give some historic perspective on the evolution of network support in Slackware, because I was involved in this a lot.

History

I have been dabbling with Slackware's network support for a long time now. My interest started when I became an IBM employee and was exposed to a Token Ring network (which is the type of network infrastructure that IBM was known for, as opposed to Ethernet which was what the rest of the world was using). Token Ring interfaces are called **tr0**, tr1, etc...
Now, you may remember if you have used Slackware long enough that the network device configuration script `/etc/rc.d/rc.inet1` had the name “`eth`” hard-coded. Slackware supported four network interfaces out of the box: they had to be called eth0, eth1, eth2 and eth3. This meant that I had to patch the script to add support for my Token Ring cards. I had to repeat this patching procedure for every computer I installed Slackware on (even at IBM, I could use Slackware because I deployed these as local servers for groups of developers in the office). At some point this became quite tedious, so I approached Pat Volkerding and asked him if he would incorporate my patch into the Slackware scripts (at the source). Pat was very friendly (of course) but also would not use my patch for quite some time (of course 🙂.

It would take as long as Slackware **10.2** to support network cards out of the box whose name did not

start with "*eth*" ...
Also in Slackware **10.2**, the capability was added to start/stop/restart every individual interface instead of the "all or nothing" approach of the earlier `rc.inet1` script. See also the section (re)starting a network interface

These patches were fairly non-intrusive, but adding support for wireless network interfaces would be a lot more complicated.
Historically, the pcmcia subsystem already supported (wired as well as) wireless cards for some time, but these cards with a 16-bit PCMCIA interface were typically configured using the `/etc/pcmcia/network.opts` and `/etc/pcmcia/wireless.opts` files. The introduction of wireless *PCCard* (which is essentially 32-bit PCI with a PCMCIA interface) and *PCI* devices demanded a new approach. Support for these cards would have to he added to the `rc.inet1` script.

The first release of Slackware to have support for *PCI* and *PCCard* wireless network cards was **10.0**.
The script `/etc/rc.d/rc.wireless` was added. It takes care of configuring the wireless parameters for a network interface. This script does not run independently (although the original version of Slackware 10.0 would not complain if you did). Instead, it is executed by the generic network configuration script `/etc/rc.d/rc.inet1`. Note that this happens for every network interface which is being initialized - wired as well as wireless. The `rc.wireless` script will return control to `rc.inet1` immediately if it determines that the interface has no wireless capabilities.
If a wireless interface is detected, `rc.wireless` will use `iwconfig`, `iwpriv` and possibly `wpa_supplicant` to associate the card with an access point (in *managed* mode) or peer it with another computer (in *ad-hoc* mode). When the script completes it's work, it hands back control to the `rc.inet1` script which continues with the (DHCP or static) IP address assignment.
If you remove the executable bit from this script, it will never be run. This can be beneficial if you have written your own wireless script and don't want Slackware to mess it up.


The `/etc/rc.d/rc.wireless` script is not meant to be run on it's own by the user!

Typically, wireless network cards are assigned a name different from **`ethN`** (`wlanN`, `raN` and `athN` are common names) Before the release of Slackware **10.2**, these cards could not easily be configured using the standard network configuration files. Support for arbitrary network interface names (other than `ethN`) was added in Slackware 10.2, which made the process of configuring wireless interfaces a lot easier.

> We are going to assume here that you are running Slackware 10.2 or newer.
> Most people with wireless cards will be better off with a recent release like Slackware 12.1 anyway.
> For older releases, there are instructions in an other Wiki article about updating the network scripts.

The first versions of the `rc.wireless` script relied on a configuration file `/etc/rc.d/rc.wireless.conf` for all of your wireless card's configuration. Starting with Slackware **10.2**, it is also possible to use the generic configuration file `/etc/rc.d/rc.inet1.conf` to store wireless parameters.
Support for WPA encryption (using wpa_supplicant) was also added, although it would take until Slackware **11.0** before a *wpa_supplicant* package was actually added to the */testing* directory. In Slackware **12.0**, wpa_supplicant finally became part of the 'N' package series.

Network subsystem

Slackware's network subsystem is straight-forward. A network device will be used by Slackware only if Slackware finds the minimally required configuration data. Most important: *it must be known if the card will use DHCP to obtain an IP address or that a static IP address will be used*. Additionally for a wireless interface it is good to have configured it's wireless parameters.
So, how does Slackware load drivers for, configure and activate a network interface?

- Automatically at boot time
  At boot time, the *UDEV* sub-system probes your hardware and based on the information it receives, loads a kernel module (or several). The feedback it receives from the kernel together with the UDEV rulesets, determine how the hardware is configured and activated. For network interfaces, the script **`/etc/udev/rules.d/90-network.rules`** determines the actions. The UDEV rule file in turn runs the script **`/lib/udev/nethelper.sh`** with the name of the interface (which was created by the kernel as the driver loaded) as a parameter. It checks on how far the boot has progressed. Activating the network must not happen too soon because some required directories (*/var* for instance) may not yet be mounted if they are not on the root partition. But ultimately, the `nethelper.sh` script will run the following command for any interface called *INTERFACE* (eth0, eth1, ath0, wlan0, you get it):

```
/etc/rc.d/rc.inet1 INTERFACE_start
```

  which will configure, then activate, the network interface with that name. But only if you added the required information to the configuration file `/etc/rc.d/rc.inet1.conf` !
  UDEV will also ensure that your network interface will always keep the name it initially got. As an example, think about what happens if you have a *`eth0`* interface and then you add an extra network card to the computer. This new card could get detected before the old one and this would result in the kernel assigning *`eth0`* to the new card, leaving *`eth1`* for the old card. To prevent this mix-up of names, UDEV uses the script **`/etc/udev/rules.d/75-persistent-net-generator.rules`** to create a file **`/etc/udev/rules.d/70-persistent-net.rules`** with lines that look like this:

```
# This file was automatically generated by the
/lib/udev/write_net_rules
# program run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single line.

# PCI device 0x8086:0x101e (e1000)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:0d:60:b1:b9:43", ATTR{type}=="1", NAME="eth0"

# PCI device 0x168c:0x1014 (ath_pci)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:05:4e:47:7c:28", ATTR{type}=="1", NAME="ath0"
```

  You see here, that a card's *MAC address* defines the interface name it will keep getting as long as this file exists. You can safely delete the file if you think there is something wrong with the naming of your network interfaces (like in the case where you cloned your Slackware system to use it on another computer or in a Virtual Machine) because UDEV will just create a new file

with correct informations the next time you boot the computer.

- Pre-defined at boot time
  If UDEV does not detect your card, or if you do not use UDEV, or if you just want to make sure that your card's driver is loaded always, you can create a file by the name of **`/etc/rc.d/rc.netdevice`** and add modprobe command(s) there that load the driver for your card. A typical modprobe line would look like this:

  ```
  /sbin/modprobe 3c59x
  ```

  Do not forget to make that file executable!

  ```
  chmod +x /etc/rc.d/rc.netdevice
  ```

  After UDEV has had it's chance to detect hardware, the Slackware initialization script **`/etc/rc.d/rc.M`** will run **`/etc/rc.d/rc.inet1`** once without parameters, ensuring that any network interface will be configured that is available to the system and for which we have configuration data, but still is in an unconfigured state (this includes the interface we created with the use of the `rc.netdevice` file).

- Runtime
  UDEV will also take care of dynamic additions and removals of hardware. This involves networking hardware as well - think of wireless PCCard or USB devices. The same basic procedure as described in the previous bullet pont applies here, with the addition that on hardware removal, ultimately the following command will run:

  ```
  /etc/rc.d/rc.inet1 INTERFACE_stop
  ```

  which will de-activate the interface.

- User induced
  You as the user can decide to stop, start or restart any network interface by directly running one of the commands

  ```
  /etc/rc.d/rc.inet1 INTERFACE_stop
  /etc/rc.d/rc.inet1 INTERFACE_start
  /etc/rc.d/rc.inet1 INTERFACE_restart
  ```

  That last command will mostly be used after changing one or more of the interface's configuration parameters like for instance the IP address, or the wireless encryption key.

Network configuration file rc.inet1.conf

- The file `/etc/rc.d/rc.inet1.conf` consists basically of a series of variable array definitions. Array elements with the same index number will all belong to the same network interface. By default, index number **0** is used for the configuration of interface **eth0**, index number **1** is used for **eth1** and so forth. The default interface name can be overruled by the use of the variable **IFNAME**.
  This is what a typical section of the file looks like - I have selected every array variable with the index number [**0**]:

```
# Config information for eth0:
IPADDR[0]=""
NETMASK[0]=""
USE_DHCP[0]=""
DHCP_HOSTNAME[0]=""
DHCP_TIMEOUT[0]=""
```

This is information which is easily editable using something like *vi* or *nano*. And indeed, this is how Slackware's network configuration is updated.

- But the initial configuration (usually during the installation of Slackware, but the command is available on the running system as well) is done by running the script

```
netconfig
```

Unfortunately, the netconfig program only cares about the interface **`eth0`**, and the questions it asks about your interface lack the one that goes "*what is the name of the interface you use?*" In most cases, *`eth0`* is indeed the network interface you are going to use. But especially where wireless cards are involved, interfaces are called anything *but* `ethN`. There is an updated version of *netconfig* in the making which supports arbitrary interfaces and wireless parameters as well, but at the time I can give no estimate when this will go into the distribution.

Rule of thumb: use `netconfig` script only for your primary wired interface configuration. For any other (including wireless) card's configuration, use `vi /etc/rc.d/rc.inet1.conf` instead

- The boot-time network configuration script `/etc/rc.d/rc.inet1` will look for up to 6 interface definitions in `rc.inet1.conf`. This is usually enough, but in case you need to expand this value, look for the line in `/etc/rc.d/rc.inet1` that says

```
MAXNICS=6
```

and change the number to a higher value.

Network configuration parameters

Here follows a list of network parameters you can set for any card. As an example I've taken the section that is usually taken for **`eth0`** i.e. the array variables with index [**`0`**]:

```
# Config information for eth0:
IPADDR[0]=""                    # Set this value to an actual IP address if
                                # you want static IP address assignment
NETMASK[0]=""                   # With a static IP address, you are required
                                # to also set a netmask (255.255.255.0 is
common)
USE_DHCP[0]="yes"               # If set to "yes", we will run a DHCP client
```

```
                                # and have the IP address dynamically
assigned
DHCP_HOSTNAME[0]="mybox"        # Tell the DHCP server what hostname to
register
DHCP_TIMEOUT[0]=15              # The default timeout for the DHCP client to
                                # wait for server response is 30 seconds,
                                # but you might want a shorter wait.
IFNAME[0]="eth0:1"              # Set up an IP alias.
HWADDR[0]="00:01:23:45:67:89"   # Overrule the card's hardware MAC address
MTU[0]=""                       # The default MTU is 1500, but you might need
                                # 1360 when you use NAT'ed IPSec traffic.
DHCP_KEEPRESOLV[0]="yes"        # If you dont want /etc/resolv.conf
overwritten
DHCP_KEEPNTP[0]="yes"           # If you don't want ntp.conf overwritten
DHCP_KEEPGW[0]="yes"            # If you don't want the DHCP server to change
                                # your default gateway
DHCP_IPADDR[0]=""               # Request a specific IP address from the DHCP
                                # server
```

If you want your Slackware system to configure a network interface at boot time, you will have to supply a value for **exactly one** of the following two variables:

```
IPADDR[0]=""   # Needs an IP address like "192.168.0.12"
USE_DHCP[0]="" # Needs the value of "yes" - anything else will
be ignored
```

If both variables are empty, the interface will not be configured and activated. If both variables have values, then the interface will be configured for DHCP

Wireless Networks

If you have a Wireless Access Point that is broadcasting its station ID (the *ESSID*), and is not configured for encrypted traffic, then you're ready to go with the default configuration as it comes with Slackware. This kind of open wireless network is typical when

1. you just unpacked your Wireless Access Point, and didn't have time yet to configure it;
2. you are at an airport/hotel/pub where they offer free wireless access.

If you need to configure specific parameters to make the wireless card talk to your Access Point - for instance, the ESSID (in case the Access Point is hiding its station ID), or the channel, or a WEP key, etc) then you will need to edit either the file

```
/etc/rc.d/rc.wireless.conf
```

or the file

```
/etc/rc.d/rc.inet1.conf
```

(one of the two will do) and add a specific configuration that matches your wireless card and Access Point.

Wireless configuration in rc.wireless.conf (deprecated)

In the previous section, I briefly mentioned the fact that you can store your wireless network parameters (excluding WPA) in both `rc.wireless.conf` and `rc.inet1.conf`. I will show you how to use `rc.wireless.conf` but first let me explain why I prefer to let people use `rc.inet1.conf` instead.

Originally, `rc.wireless.conf` was the file to store your wireless parameters. When support for wireless parameters was also added to `rc.inet1.conf` this was done with a reason.
The `rc.wireless` and `rc.wireless.conf` files were initially based on the *pcmcia* scripts for wireless cards. That means your card's parameters were tied to it's *MAC address* (with the option to use a wildcard in that MAC address to support different brands of cards). This is different from the standard way of configuring a network card in Slackware, where network configuration parameters are tied to the name of the interface. Adding the option to define your wireless configuration parameters in `rc.inet1.conf` allows you to keep all your network settings (apart from WPA) in one file: `rc.inet1.conf`. I can hear you think "*what happens if I define a wireless parameter in both files?*". Good question! In fact, a parameter value which is set in `rc.inet1.conf` will always override the value you might have set for that same parameter in `rc.wireless.conf`.
I regularly hear from Slackware users that they are confused by the possibility to use two configuration files for the same settings. Well, I agree. This `rc.wireless.conf` is a historic left-over and ultimately I would like to see it removed from the Slackware *wireless-tools* package completely. I think there is no good reason to want to keep using `rc.wireless.conf`. In fact, you can safely delete that entire file! But for those who do not want to say goodbye to it, we will keep support for it in the network configuration scripts.

Any wireless parameter defined in `rc.wireless.conf` and called **FOO** is equivalent to the wireless parameter called **WLAN_FOO[n]** in `rc.inet1.conf` (the *[n]* being the index relating to your card). The **WLAN_** prefix is what distinguishes the two.

Please use `/etc/rc.d/rc.inet1.conf` as the single configuration file for all your network parameters (wired as well as wireless)

Let us have a better look at this file anyway.

- rc.wireless.conf
  You will notice that the content of `/etc/rc.d/rc.wireless.conf` is basically a number of sections that apply to certain (ranges of) wireless network cards. The distinguishing factor is the hardware address (the *MAC address*) of a card. A section for a specific card or range of cards looks like this:

```
MAC_Address)
    INFO="a string that decribes your card type"
    PARAMETER1="value1"
    PARAMETER2="value2"
```

```
    [more parameters] .......
    ;;
```

The MAC_Address in this example can be a full MAC address (six *HEX* bytes separated by colons, like `00:12:8E:A0:32:DC`) that matches a single network card, or a wildcard address that matches a whole range of cards, typically all cards from a specific vendor (like `00:12:8E:A0:*`).
If you intend to use `rc.wireless.conf` you will probably have to add such a section for your specific card, and this is how to do it:

- If your wireless interface called *wlan0*, run `ifconfig wlan0` to get the MAC address of the card.
- Edit `/etc/rc.d/rc.wireless.conf` and comment out this section right in the beginning of the file:

```
  *)
      INFO="Any ESSID"
      ESSID="any"
      ;;
```

  so that it will look like this:

```
# *)
#     INFO="Any ESSID"
#     ESSID="any"
#     ;;
```

- Somewhere further below the lines you just commented out, add a few lines (easiest is to add it to the very bottom of the file for instance, **right above** the `esac` line) that will apply to your card, like these:

```
  00:06:25:13:2B:D4)
      INFO="D-LINK DWL-G510 revB1"
      ESSID="your_ap_essid"
      KEY="0100030203"
      ;;
```

  The first line is your card's MAC address followed by a ')', and the last line must only consist of two semicolons - all by themselves (copy and paste one of the available examples if you're unsure).
  Your MAC address, ESSID, KEY and info comment are obviously going to be different from the values in the above example.

Wireless configuration in rc.inet1.conf

- rc.inet1.conf
  We covered this file earlier on in the *wired* section of this article. For *wireless* cards, it is a matter of extending the parameter definitions of your card with the ones that relate to wireless. All these parameters (or better, *variables*) start with the prefix **WLAN_**. The `rc.inet1.conf` file which is installed by the *network-scripts* package has a lot of examples at the bottom.

Let's go through a typical configuration of a wireless interface called *ath0*.

- The first thing to do is define the interface name because it is different from the default *ethN*. The variable **IFNAME** is used to define the interface name. You will need to add or modify a few lines in `/etc/rc.d/rc.inet1.conf` in order to get a configuration like this:

```
# Config information for ath0 (using dhcp):
IFNAME[1]="ath0"
IPADDR[1]=""
NETMASK[1]=""
USE_DHCP[1]="yes"
DHCP_HOSTNAME[1]="mywirelessbox"
```

Or like this:

```
# Config information for ath0 (using static IP address):
IFNAME[1]="ath0"
IPADDR[1]="192.168.3.11"
NETMASK[1]="255.255.255.0"
USE_DHCP[1]=""
DHCP_HOSTNAME[1]=""
GATEWAY="192.168.3.1"
```

These are example values of course and you will have to substitute your own.

- NOTE
In the above example, where I used the index 1, like in: VARIABLENAME[**1**], you may use whatever index is not used for any other card that you may have installed. If you do not have an eth**0** interface for instance, you might as well want to use the unused **0** array index. The last configuration example would then look like this:

```
# Config information for ath0 (using static IP address):

IFNAME[0]="ath0"
IPADDR[0]="192.168.3.11"
NETMASK[0]="255.255.255.0"
USE_DHCP[0]=""
DHCP_HOSTNAME[0]=""
GATEWAY="192.168.3.1"
```

Obviously, any array index value ([0],[1],[2], ...) in /etc/rc.d/rc.inet1.conf should be used for exactly one card's configuration. If you copy a set of lines, be sure to change the array index to an unused value. If you forget this, and create a double entry, then Slackware will happily forget about the first, and will use only the last value for any parameter found in the file.

Keep the index value below **6** whenever possible. The `rc.inet1` script will not look for indexes larger than **6**

- In order to get your network card connected, you will have to add one or more wireless configuration parameters, like a WEP key, the ESSID, and such:

```
IFNAME[1]="ath0"
   ...
WLAN_MODE[1]=Managed
WLAN_ESSID[1]="my access point"
WLAN_KEY[1]="D5AD1F04ACF048EC2D0B1C80C7"
```

- Note that I deliberately used an ESSID (the access point's Station Set Identifier) which has spaces in it. This requires that you use quotes around the name: *“my access point”*. When your access point has a name without spaces, you do not need these quotes - in fact it is better to leave those out: *WLAN_ESSID[1]=Darkstar*.
- **NOTE about WEP encryption:**
  You may have defined your WEP key as a string of ascii characters (i.e. a readable passphrase like “Hogwarts”) instead of a string of hexadecimal characters (like “6CC07C36169B8E7524886F9A19”). If you want to use this readable string instead of typing a series of HEX characters, you can use the following key format in `rc.inet1.conf`

  ```
  WLAN_KEY[1]="s:Hogwarts"
  ```

  This is for a 128-bit (aka 104-bit) WEP key. The even weaker 64-bit (aka 40-bit) WEP keys are still being used - in this case you would need to provide one of 4 keys (or all four with one of them defined as active), this key would have to be the one that the access point considers active as well. Suppose we want to set key [2] to the ascii value “Hogwarts” and then make this the active key, this will take two `iwconfig` commands: “`iwconfig key [2] s:Hogwarts`” and “`iwconfig key [2]`”. These commands can be combined into one: “`iwconfig key [2] s:Hogwarts key [2]`” and the corresponding entry in `rc.inet1.conf` would become (the first “key” word removed):

  ```
  WLAN_KEY[1]="[2] s:Hogwarts key [2]"
  ```

  WEP key generators can be found all over the internet. A nice one is PowerDog's cgi script. Using WPA encryption is recommended, see the section that comes next if you need to know how to configure WPA encryption.

It depends on your access point and the quality of the signal (think of interference because of nearby Access Points when you live in a densely populated area) whether you have to explicitly configure parameters as the channel and the rate. Leaving these undefined will cause the driver to scan for the appropriate channel and settle for a dynamic transmission rate.

Wireless configuration parameters

A comprehensive list of supported wireless parameters follows (taken from `rc.inet1.conf`):

```
IFNAME[4]="ath0"                # Use a different interface name instead of
                                # the default 'eth4'
WLAN_ESSID[4]=DARKSTAR          # Your access point's name
```

```
WLAN_MODE[4]=Managed                # "Managed" mode for use with Access Points.
                                    # "Ad-Hoc" is for peer-to-peer connections.
WLAN_RATE[4]="54M auto"             # The transmission rates you want the driver
to try
                                    # ("auto" means that bandwidth can be
variable)
WLAN_CHANNEL[4]="auto"              # The channel to which the Access Point is
tuned
                                    # ("auto" to let the driver find out the
correct channel)
WLAN_KEY[4]="D5A31F54ACF0487C2D0B1C10D2"
                                    # Definition of a WEP key
WLAN_IWPRIV[4]="set AuthMode=WPAPSK | set EncrypType=TKIP | set
WPAPSK=thekey"
                                    # Some drivers require a private ioctl to be
                                    # set through the iwpriv command. If more
than
                                    # one is required, you can place them in the
                                    # IWPRIV parameter (separated with the pipe
(|)
                                    # character, see the example).

WLAN_WPA[4]="wpa_supplicant"        # Run wpa_supplicant for WPA support
WLAN_WPADRIVER[4]="ndiswrapper"# Tell wpa_supplicant to specifically use the
                                    # ndiswrapper driver. If you leave this empty
                                    # the 'wext' driver is used by default; most
                                    # modern wireless drivers use 'wext'
WLAN_WPAWAIT[4]=30                  # In case it takes long for the WPA
association
                                    # to finish, you can increase the wait time
before
                                    # rc.wireless decides that association failed
                                    # (defaults to 10 seconds)
```

Read the **iwconfig** man page if you want to know more about the possibilities for the various **WLAN_** parameters that are available. These parameters translate directly into `iwconfig` commands

WPA encryption

WPA stands for *Wi-Fi Protected Access*. It is an encryption standard which was devised after it became clear that the older WEP (*Wired Equivalent Privacy*) encryption algorhitm was seriously flawed - WEP can be cracked in a matter of seconds. Note that WPA is not 100% safe either, but much harder to crack than WEP. Just make sure that you use a non-trivial WPA passphrase to create your WPA key. The WPA cracking methods that rely on dictionary attacks will eventually break through your encryption if you use common words to make up your passphrase.

OK... configuring WPA encryption seems to be quite problematic for many people. But it is not hard at

all if you know *where* to configure! I am assuming that you already have your wireless card up and running without any encryption, or possibly with WEP encryption - the way to do that was shown in the previous section.
Do not try to add WPA support if you do not yet have a functional wireless network connection! It will make the toubleshooting a lot more difficult if it does not work right away.

- To enable WPA support for your card, ensure that you installed the wpa_supplicant package (it is part of a full Slackware install) and then open the file `/etc/wpa_supplicant.conf` in an editor - as root, because this file is protected from being seen by normal users. It should look something like this:

```
  ctrl_interface=/var/run/wpa_supplicant
  ctrl_interface_group=0
  eapol_version=1
  ap_scan=1
  fast_reauth=1
  network={
        scan_ssid=0
        ssid="your_essid"
        proto=WPA RSN
        key_mgmt=WPA-PSK WPA-EAP
        pairwise=CCMP TKIP
        group=CCMP TKIP
        psk=your_64_hex_characters_long_key
  }
```

  but you'll need to supply your own values for the *ssid* and the *psk*. The *ssid* is the same as the *ESSID* you use in `rc.inet1.conf`. The *psk* is the *Pre Shared Key*.

- There is a way to generate the hexadecimal value for the PSK if you have an access point which uses a passphrase. As root, run:

```
wpa_passphrase YOURSSID passphrase
```

  with the *YOURSSID* being the ESSID of your Access Point and *passphrase* is the ascii string you entered in the Access Point's *WPA-PSK* configuration section. You'll receive an output, which looks like this:

```
network={
    ssid="YOURSSID"
    #psk="passphrase"
psk=04dffae0172e3a255e5bab6f28ab78cc23d845f3dd8d4a63ba64a37555e2a33b
}
```

  Next, you should copy the three lines that you find inside the `network={}` section of the command's output and paste them inside the `network={}` section of the file `/etc/wpa_supplicant.conf`. Do not forget to check the permissions of the configuration file! The key that it contains should be protected from prying eyes.

```
chmod 600 /etc/wpa_supplicant.conf
```

- Now, if your network configuration for a wireless interface *ath0* in `rc.inet1.conf` looked like this at first:

```
# Config information for ath0 (using dhcp):
IFNAME[1]="ath0"
IPADDR[1]=""
NETMASK[1]=""
USE_DHCP[1]="yes"
DHCP_HOSTNAME[1]="mywirelessbox"
```

then you'd have to add two lines for WPA support so that it will read:

```
# Config information for ath0 (using dhcp):
IFNAME[1]="ath0"
IPADDR[1]=""
NETMASK[1]=""
USE_DHCP[1]="yes"
DHCP_HOSTNAME[1]="mywirelessbox"
WLAN_WPA[1]="wpa_supplicant"
WLAN_WPADRIVER[1]="wext"
```

The value "*wext*" for the variable *WLAN_WPADRIVER* stands for "*Wireless Extensions*". It is the protocol with which wpa_supplicant and the wireless driver communicate. *Wireless Extensions* allow for a simple framework that connects arbitrary (and well-written) wireless kernel drivers and the userland program wpa_supplicant together. Not too long ago, neither wpa_supplicant nor wireless drivers used *Wireless Extensions* by definition, so it is likely that for older releases of these sotwares, wpa_supplicant needs to talk to the driver using it's own protocol. You can easily find out what "drivers" (cards) are supported by wpa_supplicant by just running it and inspecting the output:

```
wpa_supplicant
wpa_supplicant v0.5.10
Copyright (c) 2003-2008, Jouni Malinen <j@w1.fi> and contributors

...

drivers:
  wext = Linux wireless extensions (generic)
  hostap = Host AP driver (Intersil Prism2/2.5/3)
  atmel = ATMEL AT76C5XXx (USB, PCMCIA)
  ndiswrapper = Linux ndiswrapper
  ipw = Intel ipw2100/2200 driver (old; use wext with Linux 2.6.13 or
newer)
  wired = wpa_supplicant wired Ethernet driver

...
```

For instance: If you use an older version of ndiswrapper you may have to use "ndiswrapper" instead of "`wext`" as the driver (note that current releases of ndiswrapper require "`wext`")

```
WLAN_WPADRIVER[1]="ndiswrapper"
```

- After making the required changes to the configuration files, you can restart your wireless network card (`ath0`) now, by running

```
/etc/rc.d/rc.inet1 ath0_restart
```

Slackware will tell *wpa_supplicant* to use the "`wext`" protocol if you do not specify a driver yourself. So, basically the line

```
WLAN_WPADRIVER[1]="wext"
```

can safely be omitted.

WPA2

WPA2 is considered a safer encryption protocol than WPA. However, not all (older) wireless access points support it because of the greater processing power the WPA2 protocol demands for the packet encryption/decryption.

* The wpa_supplicant.conf example in the previous section will support WPA as well as **WPA2** encrypted networks. In the following line taken from `wpa_supplicant.conf`

```
  proto=WPA RSN
```

the string *WPA2* is an alias for *RSN*, so that that line can be written like this as well:

```
  proto=WPA WPA2
```

* WPA2 support for the legacy RaLink drivers by serialmonkey is configured with an "`iwpriv`" command like this (check the earlier section about rc.inet1.conf to see the difference with the WPA1 example given there):

```
WLAN_IWPRIV[?]="set AuthMode=WPA2PSK | set EncrypType=AES | set
WPAPSK=the_64_character_key"
```

WPA debugging

If your WPA connection does not activate, there are several ways to debug the problem. For the sake of the examples I will assume that the name of your wireless interface is **`ath0`** - substitute your own card's name if you need to apply these commands.

- Debug the WPA authentication process.
  Make sure the network interface is down (run

```
/etc/rc.d/rc.inet1 ath0_stop
```

to make sure). Start the wpa_supplicant daemon as a foreground process with additional debugging enabled:

```
wpa_supplicant -dw -c/etc/wpa_supplicant.conf -Dwext -iath0
```

Then activate the network interface in another terminal. Run

```
/etc/rc.d/rc.inet1 ath0_start
```

Look at the output of wpa_supplicant in the first terminal, it might give you pointers to look for a solution.

- Get a run-time status overview of the supplicant:
  As root, run

```
wpa_cli status
```

to see the current status of wpa_supplicant's authentication process. If you have more than one wireless interface that uses wpa_supplicant, you will have to specify the interface you want to query:

```
wpa_cli -i ath0 status
```

- Debug Slackwares network initialization.
  Change

```
DEBUG_ETH_UP="no"
```

to

```
DEBUG_ETH_UP="yes"
```

in the file `/etc/rc.d/rc.inet1.conf` and look for *logger* messages that are being written to `/var/log/messages` while the interfaces are configured. Maybe those messages will help you trace your problem.
NOTE: with debugging enabled, Slackware will write your WEP/WPA keys to the message log as well, in clear text!

- The WPA association might take a long time.
  Start the interface again after a little time, this may help if it takes wpa_supplicant a long time to associate (no *restart*, just a *start*):

```
/etc/rc.d/rc.inet1 ath0_start
```

This “start” command will leave the still running wpa_supplicant process alone, so that a possibly delayed association to an Access Point is not disrupted. If this makes your wirelesss work, but the problem occurs quite often when you don't run this extra manual command, you

can change the 'wait' time for the WPA authentication process by editing the file `/etc/rc.d/rc.inet1.conf` and adding the line

```
WLAN_WPAWAIT[?]=30
```

or any other larger value that helps your particular setup.
NOTE: in the last line make sure that you replace the questionmark in **[?]** with the array value that matches your wireless card configuration In the above example this array index would be [**1**] and the actual line in `rc.inet1.conf` would look like

```
WLAN_WPAWAIT[1]=30
```

If you set **DEBUG_ETH_UP="yes"** then you will notice messages in your "`/var/log/messages`" file like this:

```
WPA authentication did not complete, try running '/etc/rc.d/rc.inet1
ath0_start' in a few seconds.
```

- The Access Point is not broadcasting the SSID.
  Some succeed, some fail in getting WPA to work when the Access Point has a *hidden SSID*. Check if your AP is broadcasting the SSID and if not, enable it. There is little point in hiding the SSID anyway, because even with network encryption there will always be packets that need to be transfered in the clear. One of the things that can not be encrypted is the ESSID! How else can a wireless card find out that it talks to the right Access Point if it cannot read the ESSID. Anyway, with WPA as a protection layer you should not be afraid of break-ins (as long as you do not use easy-to-guess passphrases!!!

(Re) starting a network interface

In Slackware, the way to start your network (the configuration of your *nics* and bringing the interfaces up, and creating a default route if required) is by running the command

```
/etc/rc.d/rc.inet1
```

Restarting the whole network is done in a similar fashion:

```
/etc/rc.d/rc.inet1 restart
```

This is quite crude, and not adequate for the dynamic detection and configuration of network devices. Therefore, when your computer boots, and UDEV detects your network hardware, it will run the following command after loading the kernel driver and determining the name of the interface (let's assume that it is *wlan0*):

```
/etc/rc.d/rc.inet1 wlan0_start
```

More generically speaking, you can start/stop/restart any network interface yourself by running one of the commands

```
/etc/rc.d/rc.inet1 INTERFACE_start
```

```
/etc/rc.d/rc.inet1 INTERFACE_stop
/etc/rc.d/rc.inet1 INTERFACE_restart
```

Alternative network managers

The information presented in the article up to here is historical and provides you with in-depth information about how Slackware itself can manage your computer's network configuration.
However, with the advance of mobile computers and graphical desktops, alternative means of managing network connectivity have been developed which allow for seamless roaming, VPN support and other complex scenarios.
Several alternative network managers have been added to Slackware over time, and these come with graphical front-end programs. This section of the article on networking in Slackware deals with the alternatives.

networkmanager

The Networkmanager was added to Slackware 14.0. Originally developed by Red Hat, it is now hosted by the GNOME project and has been adopted by virtually all Linux distributions.
NetworkManager is able to switch automatically between wired and wireless networks, allows VPN connections of various types and supports modems.

Read more about NetworkManager on is homepage https://wiki.gnome.org/Projects/NetworkManager

The `NetworkManager` package installs a daemon which talks to your computer's *dbus* messagebus to detect network connects/disconnects. The daemon is started at boot by making its *rc script* executable:

```
chmod +x /etc/rc.d/rc.networkmanager
```

Configuration of your wireless as well as wired interfaces is done via a client program. You can either use the GTK-based graphical `network-manager-applet` in your X Window session (KDE, XFCE, blackbox, ...), or use the *text user interface* program `nmtui` if you are not using X. If you are running KDE 4 as your Desktop Environment, then the package `plasma5-nm` will show a system tray widget. If you are running Plasma 5 Desktop Environment, then `plasma5-nm` installs a Plasma widget for the graphical management of NetworkManager. To enable that widget, right-click on the system tray and select *add widgets*, then search for *network* and drag the widget to your system tray. Once the widget is visible in your Plasma system tray you can use it to interact with the daemon.

If you want to use NetworkManager, you will have to remove any network interface configuration information from `/etc/rc.d/rc.inet1.conf` in order to prevent a struggle for power between NetworkManager and Slackware's `rc.inet1` script.

wicd

Wicd (pronounced as *wicked*) aims to provide a simple interface to connect to networks with a wide

variety of settings. Some of Wicd's features include:

- Ability to connect to wired and wireless networks
- Profiles for each wireless network and wired network
- Many encryption schemes, some of which include WEP/WPA/WPA2
- Remains compatible with wireless-tools
- Tray icon showing network activity and signal strength

Read more about it here: http://wicd.net/

You can find the `wicd` package in the `/extra` section of the Slackware distribution. It is not installed by default as part of a full installation.
Wicd installs a daemon which talks to your computer's *dbus* messagebus to detect network connects/disconnects. The daemon is started at boot by making its *rc script* executable:

```
chmod +x /etc/rc.d/rc.wicd
```

Configuration of your wireless as well as wired interfaces is done via a *wicd client*. You can either run the graphical *wicd-client* in your X Window session (KDE, XFCE, blackbox, ...), or use the console program *wicd-curses* if you are not using X. If you are running KDE4 as your Desktop Environment, then the package `wicd-kde` installs a KDE widget for the graphical management of your wicd daemon. To enable the `wicd` widget, right-click on the system tray and select *add widgets*, then search for *wicd* and drag the widget to your system tray. Once the widget is visible in your KDE system tray you can use it to interact with the daemon.

If you want to use wicd, you will have to remove any network interface configuration information from `/etc/rc.d/rc.inet1.conf` in order to prevent a struggle for power between wicd and Slackware's `rc.inet1` script.

lxnm

wifi-radar

knetworkmanager

Do not use this. The KDE tools are not compatible with Slackware.