Linux links wirelessly

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Tutorial overview, background, and assumptions

Tutorial overview

This tutorial is targeted at developers, system administrators, and end users seeking a detailed, step-by-step guide to configuring a wireless network card under Linux. After working through the material presented, the reader should come away from this tutorial enriched with the following:

- A "broad strokes" understanding of the overall process involved in configuring a wireless card under Linux, and why the author uses the detailed procedure he does
- An understanding of how to adapt the examples presented to your own unique requirements
- How to recompile your kernel to support the PCMCIA-CS library
- How to fetch and compile the PCMCIA-CS source
- How to find and compile the requisite drivers to support your chosen wireless card (if required)
- And finally, how to configure your wireless card to connect to a Wireless Access Point (WAP)

Broad strokes

For many users, the aspect of choice is one of the fundamental draws to open source offerings. But there is a downside to multiple-choice offerings, especially for those coming from a background of "plug-and-play" and the promise of inserting your card/device and having the operating system "automagically" load and configure your system for immediate use. Linux does not work that way for the most part. Linux is designed from the ground up to be as extensible as possible. To accomplish this, developers have historically relied heavily on user participation throughout the configuration process.

The number of wireless network cards supported by Linux has increased dramatically over the last year or two. For some devices, configuring wireless connectivity can be as simple as procuring a copy of the latest release of your preferred distribution, inserting your wireless card, clicking through the setup process, and entering the correct networking parameters. But if your distribution fails to detect your wireless card or detects it incorrectly, chances are good you'll be left without connectivity and wondering which direction to head off in next. That's where this tutorial comes into play. Hopefully the background and procedures outlined here will empower you with the necessary skills to troubleshoot your installation and get it running without major headaches.

As you read through the material contained in this tutorial, please keep one thing in mind: the procedures described here are not "gospel." Instead, they serve as a framework; a framework the author has used "in the real world," with a wide variety of systems, with relatively consistent results.

Procedural overview

The procedural framework alluded to in the previous panel is comprised of the following steps or tasks:

1. Recompile your kernel and remove any reference to the kernel-supplied PCMCIA drivers.
2. Download and compile the PCMCIA-CS libraries and drivers.
3. Fetch, compile, and install any required drivers for your wireless card. This step will be optional for some, and mandatory for others. Which camp you fall into depends in large part on your choice of wireless network cards.
4. Configure your wireless card to communicate with your WAP.

Please note: The aboves steps *must* be completed in the order outlined. The kernel will, by default, use its own PCMCIA/wireless drivers unless you specifically tell it to do otherwise; your system will not use the PCMCIA-CS libraries and drivers until you "de-couple" the default kernel drivers and install the PCMCIA-CS source; and most 3rd party device drives will not compile/install without a copy of the PCMCIA-CS libraries installed on your system.

Prerequisites and assumptions

To achieve maximum benefit from the concepts, procedures, and examples presented in this tutorial, the reader should have a good grasp of basic *NIX administrative tasks including program installation, filesystem layout, moving and copying files, file permissions, and editing system configuration files. Several good "primers" on system administration are listed in Resources on page 20 at the end of this tutorial.

In additon:

- It's assumed you have access to a "working" Linux installation, meaning the operating system is installed and configured correctly, and all basic services are functioning as advertised.
- It is also assumed you have access to a fully functional WAP, and that it, too, is configured correctly and working as it should. For details on hub/router/WAP configuration, please see the documentation for your particular device (now might also be a good time to make sure your WAP's firmware is current, and if not, apply any available updates; again, see the documentation accompanying your product for procedural specfics).
- You'll need to know how your network is currently configured, including how the system you plan on configuring acquires its IP address (static or dynamic?), the network netmask in use, your WAP's assigned SSID (similar in concept to a machine's hostname), and the channel your WAP is "listening" on.
- It's crucial you know who supplies the chipset for the wireless card you intend to configure. Keep in mind that it's not unusual for a vendor to use different chipset suppliers across a product line. In other words, just because your old wireless card used a *wlan*-based chipset, don't assume the new card you bought last week (from the same vendor) uses the same chipset. Generally speaking, every unique chipset is designed to be used with a specific driver. And while some drivers are designed to be used across a spectrum of devices and manufacturers, marrying up the wrong driver with the wrong device is a sure-fire way to promote premature hair loss. Research your wireless adapter on the Web (*Google* is your friend), check the vendor's Web site... do whatever you need to do. Just be *sure* your know beyond a shadow of a doubt both the underlying chipset, *and* which driver the manufacturer of the card recommends.

General application notes

Every effort possible was made to keep this tutorial as distribution agnostic as possible. Kernel releases and source code versions used in the examples provided are noted, and are current as of the time of this writing. Be sure to check the various source URLs listed in the Resources on page 20 section for the latest revisions, and always use the latest stable code release whenever possible. Open source projects are under constant development. Using the latest stable release, as a rule, means you'll be working with the most mature, bug-free codebase currently available.

Unfortunately, device drivers tend to be device specific. The examples in this tutorial are based on installing and configuring the latest release of the *wlan-ng* device driver. This particular driver was chosen because:

- It's a relatively common driver used by several well-know wireless cards
- The latest PCMCIA-CS release does not contain support for the *wlan-ng* driver, which means adding it to the 'CS' libraries (how to accomplish this is fully detailed in Third party device drivers on page 11)
- The wireless card the author uses on a daily basis uses the *wlan-ng* driver, which means he has lots of "in the trenches" experience compiling and configuring this particular device

About the author

Tom Syroid is a contract writer for *Studio B Productions*, a literary agency based in Indianapolis, IN specializing in computer-oriented publications. Topics of interest/specialty include *NIX system security, Samba, Apache, and Web database applications based on PHP and MySQL. He has experience administering and maintaining a diverse range of operating systems including Linux (Red Hat, OpenLinux, Mandrake, Slackware, Gentoo), Windows (95, 98, NT, 2000, and XP), and AIX (4.3.3 and 5.1). He is also the co-author of *Outlook 2000 in a Nutshell* (O'Reilly & Associates) and *OpenLinux Secrets* (Hungry Minds). Tom lives in Saskatoon, Saskatchewan with his wife and two children. Hobbies include breaking perfectly good computer installations and then figuring out how to fix them, gardening, reading, and building complex structures out of Lego with his kids.

Questions, comments, and errata submissions are welcome; you can either e-mail the author directly (*dwcomments@syroidmanor.com*) or use the Feedback on page 21 form at the end of the tutorial.

Section 2. Recompiling the kernel

Why recompile?

This section details the process of recompiling your kernel to remove the default Linux card services support; the next section covers the procedures required to reintroduce an outside source of card services to replace the kernel-based defaults.

So why remove the default kernel card services? There are several reasons:

- Generally speaking, the PCMCIA-CS card service libraries and drivers tend to be more stable, and somewhat more "well-mannered" than the kernel-based libraries when it comes to reacting with other programs and network services.
- The PCMCIA-CS source supports a wider range of cards.
- The device drivers bundled with the PCMCIA-CS source are, as a rule, updated and improved quicker than the drivers bundled with the kernel.
- And as noted in the introduction, if you need to build/install a third party driver, that driver will, in most circumstances, require that the PCMCIA-CS libraries be installed and configured.

The process of recompiling your kernel is not as difficult as it's often made out to be, and if you follow the guidelines presented in the next few panels, there's nothing you can "break" that can't be corrected or reversed. Don't, however, be discouraged if it takes several attempts to build a working, bootable kernel. The procedure is relatively simple, but it does take some familiarity with the various options available before you routinely get things right.

Overview and cautions

The process of recompiling your kernel can be broken down into the following systematic steps:

1. Ensure you have a copy of the kernel source on your system.
2. If you're recompiling an existing kernel, and you want to retain your configuration settings, be sure to copy the kernel configuration file (typically, */usr/src/linux/.config*) to a safe place. Failure to do so means you risk overwriting your current configuration.
3. CD to the top of the source tree (**`cd /usr/src/linux`**) and type **`make mrproper && make clean`**.
4. Configure your kernel. Depending on your interface preference, this is accomplished by typing **`make config`**, or **`make menuconfig`**, or **`make xconfig`** at the command line (the latter must be done from an open console window).
5. To compile the kernel, type **`make bzImage`** (or on some systems, **`make install`**).
6. The command **`make modules`** builds any modules required by the kernel; **`make modules_install`** installs them.
7. Finally, reboot your system.

The balance of this section will be spent breaking down and detailing the above steps.

Fetching and unarchiving the kernel source

First, check to see if you have a copy of the kernel source already installed on your system. Some distributions install it automatically, others do not, which simply means you'll have to find and install the correct package from your installation CDs (the kernel source package is typically named `kernel-source-2.X.XX-XX`, with the Xs denoting the kernel version and build).

Let's assume for the moment you've decided to take this opportunity and upgrade your kernel. Go to the Linux Kernel Archives Web site (see Resources on page 20 for a link), locate the requisite *tbz* file, and download it to a directory of your choosing. Next, copy the tarball to */usr/src* and change to this directory. Depending on how your directory permissions are set, you may have to be the "root" (or the "superuser") to do this. Now would be a good time to do an `ll` or "long listing." Doing so allows you to view the file naming conventions in use, and what symlinks are in place. *Always* be sure you complete this step before uncompressing a kernel source file. Some kernel source packages, when extracted, dump everything into */usr/src/linux*. If */usr/src/linux* already exists as a symlink to `linux-2.4.XX`, this effectively overwrites your current kernel source tree. In short, make it a habit to remove all symlinks from the */usr/src* directory before extracting any new kernel source there. Once the extraction process is complete, recreate the symlink (**`ln -sf /usr/src/linux-2.4.XX linux`**).

A couple of notes on upgrading the kernel

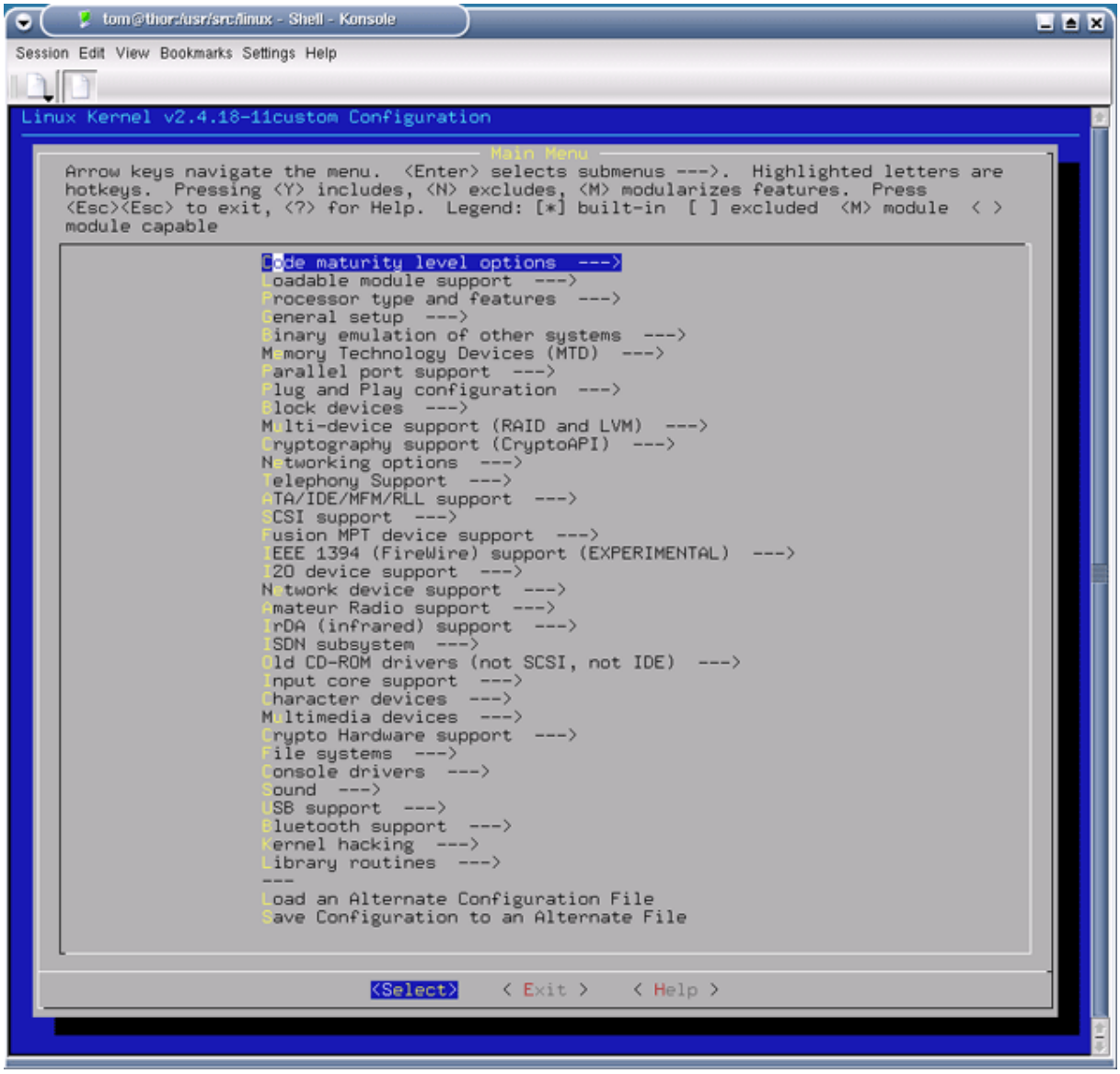
Two more important points to note: One, before embarking on any kind of kernel configuration adventures, always ensure you have a working emergency boot disk or some other means of getting back into your system should your re-compilation efforts go awry. Two, save a copy of your existing kernel configuration (if you have an existing configuration, that is) somewhere safe. For example: **`cp /usr/src/linux/.config /root/kernel-20020826`**. Next, type **`make mrproper`**. This command overwrites `.config` if it exists, and rebuilds all the kernel symbol files. Finally, type **`make clean`** to clean up any residual object files hanging around from previous kernel compilations. If you want to reuse an existing kernel configuration, copy it back to */usr/src/linux* now (using the preceding example, **`cp /root/kernel-20020826 /usr/src/linux/.config`**).

Configuring your kernel

Now we tackle the configuring of the kernel itself. Depending on personal preference, type one of the following commands:

- **`make config`**
- **`make menuconfig`**
- **`make xcoonfig`**

Each of the above methods provides an interface for configuring the kernel. The first is a simple, text-based, sequential script, the second invokes a menu-based interface, and the third opens an interface design to run in an X Window environment. The screenshot below shows the Main Menu from a `make menuconfig` command run from a KDE Konsole window.



Removing the kernel-based card services

Two changes are required to remove card services from the kernel. The first option relates to all card services (wireless and wired LAN PCMCIA adapters); the second relates only to wireless cards.

1. From the Main menu go to General setup --> PCMCIA/CardBus support. Change the PCMCIA/CardBus support option from "yes" to "no" (or from <M>/<*> to < >).
2. Next, go to Network device support --> Wireless LAN (non-hamradio). Leave the Wireless LAN (non-hamradio) option selected, but go down the list and deselect all the drivers listed. While this step is not really necessary, there's no point in building a bunch of modules we don't need and the kernel can't load. The first will be satisfied by installing the PCMCIA-CS libraries and related drivers, the second because we disabled the mechanism the kernel uses to load any card service-based drivers in the first step above.

Kernel reconfiguration, as far as card services goes, is complete. If these are the only

changes you want to make, escape out of the configuration program, and save your new configuration file. If you have more tweaking and tuning to do, continue on with the process until you're satisfied; then exit saving the configuration file on your way out.

A side-note on kernel configuration in general: If you find a configuration option you're not familiar with, select the item and click on (or "right-arrow to") the Help button. This takes you to an explanation of the option. As a general rule of thumb, if you don't know what an option does, and the help screen doesn't provide any further insight, *leave it alone*. For further information on kernel configuration and what is sometime affectionately called "performance tuning," see the Resources on page 20 section.

Building the kernel

With the kernel configuration phase complete, it's now time to build the kernel. This is accomplished with the three previously outlined commands:

- **`make bzImage`**
- **`make modules`**
- **`make modules_install`**

If any errors arise during the build process (warnings are acceptable; any line with the word "error" in it is not), don't panic (even seasoned kernel hackers screw up a kernel configuration on occasion). First, read the error carefully. It will often provide valuable clues as to the source. If the problem lies in a specific driver/module, go back, configure it out, and try again. If you get "unresolved symbol" errors, you probably forgot to run **`make mrproper`**. Go back and try again. If you find yourself really and truly stuck, try a Google search on the error, read through the FAQ available on kernel.org, or ask someone who knows more than you did.

Once you have a clean build, move or copy the file `system.map` to the */boot* directory, appending the kernel version to the end:

```
cp /usr/src/linux/System.map /boot/System.map-2.4.19
```

```
cp /usr/src/linux/arch/i386/boot/bzImage /boot/bzImage-2.4.19
```

Red Hat users have an additional step to complete: creating a new `initrd` image file (if you left the initrd option enabled in the kernel, that is). See the kernel compiling HOWTO on the Red Hat site for details (see Resources on page 20 for a link).

Two more small steps remain: Reconfigure your boot loader (LILO or GRUB) and then reboot your system to test your new kernel. The next section details how to fetch, build, and install the PCMCIA-CS card services.

Section 3. Installing PCMCIA-CS

PCMCIA-CS installation overview

Once you've successfully configured and built a kernel, installing the sources for PCMCIA-CS is straightforward. The procedure is similar for almost all device driver packages:

1. Download and unpack the source files. In the case of PCMCIA-CS, most third party drivers look for the package under */usr/src/pcmcia-cs-X.X.X* so this is the recommended location.
2. CD into the pcmcia-cs-* tree.
3. Carefully read through any provided README or INSTALL files. These are an invaluable source for installation tips and tricks, cards supported, last minute changes, etc.
4. Type **`make config`** or **`./Configure`** (note the capital 'C') and answer the questions displayed.
5. Type **`make all`**, followed by `make install`.

Configuring the PCMCIA-CS source

Unlike most programs, the PCMCIA-CS configuration process is script-based. Running **`make config`** produces a series of queries which you answer by providing a "y" (yes) or "n" (no) to the question, or in some cases, by entering the full path to a program or code source.

For most installations, the default response (shown in square brackets) is correct. For details about each option, see the PCMCIA-HOWTO file located in the */usr/src/pcmcia-cs-3.?.?* directory.

One option to pay attention to is the question "Build 'trusting' versions of card utilities". Answering "no" to this question means only root can run the bundled card services utilities. If you're the only person using the system, or you trust all users on the system, you might want to answer "yes" here.

When the script completes, a summary is displayed similar to the following:

```
Kernel configuration options:
    Kernel-tree PCMCIA support is disabled.
    Symmetric multiprocessing support is disabled.
    PCI BIOS support is enabled.
    Power management (APM) support is enabled.
    SCSI support is disabled.
    IEEE 1394 (FireWire) support is disabled.
    Networking support is enabled.
     Radio network interface support is disabled.
     Token Ring device support is disabled.
     Fast switching is disabled.
     Frame Diverter is enabled.
    Module version checking is enabled.
    Kernel debugging support is disabled.
    Preemptive kernel patch is enabled.
    /proc filesystem support is enabled.

It doesn't look like you are using 'lilo'.
```

```
It looks like you have a System V init file setup.

X Window System include files found.
Forms library not installed.
    If you wish to build the 'cardinfo' control panel, you need the Forms
    library and the X Window System include files.  See the HOWTO for details.

Configuration successful.
```

First and foremost, ensure you see the line "Configuration successful" at the end of the summary. If an error is displayed, go back and fix it before proceeding. Second, take a minute to read through the Kernel configuration options listed. If something is amiss or not as you'd like it, now's the time to fix it. As the title implies, the options shown are applicable to the current running kernel. To change a feature or option, you'll need to reconfigure and recompile the kernel.

Installing the PCMCIA-CS drivers & utilities

When you're satisfied with the configuration, go ahead and build the PCMCIA-CS libraries and drivers by typing **`make all`**. If everything compiles without error, install the product using the command **`make install`**.

If the PCMCIA-CS version you installed supports your particular wireless card (see the file */usr/src/pcmcia-cs-?* for a current list), you're all set. Skip ahead to the Network configuration on page 15 section where the configuration process is detailed. If your wireless card is not supported, you're not done yet. The next section discusses how to locate, build, and install third party PCMCIA device drivers.

Section 4. Third party device drivers

Introduction: Third party device drivers

Here's where things get "interesting" (also known in admin circles as a "multidimensional" problem). Let's say you've reconfigured your kernel, built and installed the PCMCIA-CS source as outlined in the previous section, stumbled through the configuration process (see the Network configuration on page 15 section), and your wireless card *still* doesn't work. What's the problem? Well, it could be one of several things.

The problem is, as noted in the Introduction, device drivers have an annoying habit of being *device specific*. And to further complicate the issue, hardware vendors have several bad habits of their own, including:

- Routinely changing the chipsets a given product is based on (no doubt due to the economics of supply and demand)
- Changing chipsets across product revisions (usually to take advantage of a new chipset's featureset)
- Being obtuse about detailing which chipset a given product/revision uses

For example, what's the difference between a Linksys WPC11 wireless card across versions 2.0, 2.5, and 3.0? The answer is both the chipsets used, *and* the way a given chipset identifies itself to the card services manager. To fully understand the issues involved here, it's important to understand exacty how a Linux card services driver works, which is the topic of the next panel.

Card services management 101

The card services manager is an important component of all card services applications, whether they're kernel-based, PCMCIA-CS-based, or developed by an independent, propriety vendor. Basically, the card services manager functions much like a traffic cop. The manager is typically loaded during the system boot process, and monitors all activity originating from the PCMCIA bus. When a card is inserted, a signal is sent to the card manager, which in turn returns a query to the card asking it for information which will help the manager marry up the device with the correct device driver. As you can see, this process can easily turn into a hornet's nest of potential problems. For example, the card can identify itself incorrectly (in the eyes of the card services manager); the card can identify itself in a way the card manager doesn't recognize; or the card can announce itself as a particular chipset, which the card manager incorrectly associates with the wrong device driver.

A "real world" example

Here's a "real world" example to illustrate the potential confusion that can arise between the card services manager and a specific device. I own a Linksys WPC11 Version 3.0 wireless network card. Inserting this card under the Linux kernel-based card services (kernel 2.4.18 and 2.4.19) causes the card manager to lock. The only way to fix the problem is to kill the card service process. Inserting the same card that's PCMCIA-CS-enabled causes the card manager to load the *orinco-cs* device driver, which in turn does absolutely nothing -- no lights on the card, no contact with the configured WAP, nothing. Evidently there's a problem

here somewhere, but what exactly is it? To make a long story short, both card service managers try to load an *orinco-cs* device driver, which is the wrong driver for the card. How did I figure this out? Several hours of simple trial and error: experimentation with various PCMCIA-CS versions, lots of Googling, and experimentation with various third party device drivers (if you're curious, the correct driver for the card is the *linux-wlan-ng* driver distributed by Absolute Systems; more on this in A working example (linux-wlan-ng) on page 13).

The bottom line is if your card doesn't work with either the kernel-based card service or the PCMCIA-CS offerings, your only recourse is research and trial-and-error. Given the nature of the beast, there are no one-size-fits-all solutions to the problem. There are, however, specific steps you can take to lessen the frustration and speed the time it takes to find a workable solution.

Guidelines for petulant devices

Device-specific installation/configuration guidelines are not going to help the average user get their wireless card running; unless, of course, that person happens to own the exact device being discussed. To work around this somewhat thorny dilemma, read through the guidelines/suggestions listed below. While it doesn't address locating, installing, and configuring a device driver for any one specific wireless or LAN device, it should get you pointed in the right direction.

- Check the hardware vendor's Web site. As noted, some sites contain more information than others, but it's a good place to start. If there are no hard-core device specifications listed, try sending an e-mail to the vendor's support address. Detail precisely what information you're looking for and why you need it. And don't forget to check your vendor's FTP archives for available Linux device drivers and/or HOWTOs. You'd be surprised how many vendors offer unsupported Linux drivers without advertising the fact.
- Spend some time scouring the Internet with a good search engine. Search on the obvious first (for example, your wireless card product number and revision), but don't forget to try the less than obvious: your notebook brand/model plus the words "wireless networking", the chipset manufacturer (if you know it), etc.
- Check your Linux distribution's Web site for a "miscellaneous" or "driver downloads" section. Many Linux vendors support a wide variety of devices that aren't bundled on the setup CDs for whatever reason. Alternately, contact the distribution's support group and ask if they know where you can find a driver for your device. If there's enough public demand for a given product, chances are the vendor will eventually support it in future releases.
- Once you know the chipset, and have located a potential driver, read ALL the documentation contained in the driver download package. Then search the Internet again using a handful of driver-specific keywords.
- If you have problems installing the driver under your current version of PCMCIA-CS, try going back a version or two. You might be surprised at the results.
- If you have problems installing the driver under different versions of PCMCIA-CS, look for an older version of the driver package and try that.
- Join a mailing list that targets the device driver you're having trouble with.
- Finally, if all else fails, contact the developer who maintains the device driver. Developers tend to be pretty busy people, so be sure to provide specifics to the problem you're experiencing, as well as a list of all the various options you've tried.

A working example (*linux-wlan-ng*)

We'll wrap up this section with a detailed working example of how to install a third party device driver, namely the *linux-wlan-ng* device maintained and distributed by Absolute Systems. Keep in mind that the instructions provided are specific to this particular device driver.

1. Download the latest stable *wlan-ng* package (see the Resources on page 20 section for a URL), and untar the package to a directory of your choice by typing **`tar zxvf linux-wlan-ng-X.Y.Z.tar.gz`**
2. Make sure you have kernel sources and the PCMCIA-CS sources available on your system and that you know the full path to their location.
3. CD into the *wlan-ng* directory tree. Clean up any unwanted files accidentally included in the tar package by running **`make clean`**.
4. Configure the *linux-wlan-ng* package by running the command **`make config`**. You'll be presented with a series of questions shown in the next panel.

The *linux-wlan-ng* configuration listing

Below is the *linux-wlan-ng* configuration listing presented after running the **`make config`** command as described in the previous panel. The default answer is in square brackets; to select the default, simply press `Enter`:

```
- "Build Prism2.x PCMCIA Card Services (_cs) driver? (y/n) [y]: "
     Select "y" if you want to build the Prism PCMCIA driver.
     If you select "n", the PCMCIA related questions below
     will not be asked.

- Build Prism2 PLX9052 based PCI (_plx) adapter driver? (y/n) [y]:
     Select "y" if you want to build the Prism driver for
     PLX PCI9052 PCI/PCMCIA adapter based solutions.

- Build Prism2.5 native PCI (_pci) driver? (y/n) [y]:
     Select "y" if you want to build the Prism driver for
     Prism2.5 ISL3874 based native PCI cards.  This includes
     PCI add-in cards and the mini-pci modules included in some
     notebook computers (but not all, some use internal USB modules).

- Build Prism2.5 USB (_usb) driver? (y/n) [y]:
     Select "y" if you want to build the Prism driver for
     Prism2.5 ISL3873 based USB adapters.  This includes
     USB add-on modules and the internal modules included in some
     notebook computers.

- Linux source directory [/usr/src/linux]:
     The config script will attempt to automagically find your kernel
     source directory.  If found, the kernel source source directory
     will be presented as the default selection.  If the default
     selection is wrong, you may correct it here.

- pcmcia-cs source dir [/usr/src/pcmcia-cs-3.1.29]:
     If the "_cs" driver is selected above, the configure script will
     attempt to present a reasonable default for the pcmcia source
     directory.  If the presented directory is incorrect, you may
```

```
      change it here.  If the "_cs" driver is not selected, this
      prompt will not appear.

  - Build for Kernel PCMCIA? (y/n) [n]:
      If the "_cs" driver is selected, we need to know if you are
      intending to use the kernel pcmcia code rather than pcmcia_cs.
      This prompt will not appear if the "_cs" driver is not selected.
      NOTE: AVS in-house testing is done almost exclusively using the
            pcmcia_cs package with kernel pcmcia disabled.  Therefore,
            your results with kernel pcmcia may vary.

  - PCMCIA script directory [/etc/pcmcia]:
      If the "_cs" driver is selected, this prompt allows you to
      change the location where the pcmcia scripts will be installed.
      Only do this if you have installed the rest of the pcmcia_cs
      scripts to a non-default location.

  - Alternate target install root directory on host []:
      This prompt allows you to specify an alternative root directory
      for the install process.

  - Module install directory [/lib/modules/2.2.20]:
      Select where you want the driver modules to be installed.  The
      script constructs a default location using the output of uname.
      If you have not yet installed the kernel you will run linux-wlan
      with, and the new kernel has a different version string, you will
      need to change this value.

  - Target Architecture? (i386, ppc, or alpha) [i386]:
      On some targets, we can't identify the target processor from
      compiler settings or predefined symbols.  Therefore, we need an
      explicit setting that identifies the target CPU.

  - Prefix for build host compiler? (rarely needed) []:
      When cross-compiling or using different compilers for kernel and
      user-mode software, it is sometimes (but rarely) necessary to
      specify a different compiler prefix to use when compiling the
      _tools_ that are built to run on the build host during the
      linux-wlan-ng build process.

  - Compiling with a cross compiler? (y/n) [n]:
      If you are cross compiling, we need to enable the building of
      the linux-wlan-ng build-time tools using the local host compiler
      rather than the cross-compiler.

  - Build for debugging (see doc/config.debug) (y/n) [y]:
      This option enables the inclusion of debug output generating
      statements in the driver code.  Note that enabling those statements
      requires the inclusion of insmod/modprobe command line arguments
      when loading the modules.  See the document doc/config.debug
      for more information.
```

With the configuration complete, run **`make all`** to build the package, and **`make install`** (as root) to install the drivers.

The final section of this tutorial discusses the all-important step of driver/network configuration.

Section 5. Network configuration

Configuration overview

This final section contains step-by-step instructions for configuring the network aspects of your wireless card so that it can find and connect to a wireless access point. If you've been following along with the examples provided, and have the proper device driver in place, the configuration process involves simply editing one or two files and filling in the correct network information.

All PCMCIA-related configuration files are located under */etc/pcmcia*. Note that you'll need root access to alter any files contained in this directory.

Most configuration files discussed in this section follow the same general format: a "default" options file (for example, `network`; note the lack of any file extension), and a "user-configurable" options file (for example, *network.opts*; note the distinguishing *.opts* file extension). As a general rule, any configuration files located in this directory should **not** be edited directly; leave this chore to system-level updates or the package installer. For example, extension-less configuration files are typically updated when a new PCMCIA driver is installed, the kernel is updated, or the PCMCIA-CS package is revised.

Understanding the PCMCIA configuration process

Understanding how a PCMCIA device (including most wireless devices) is initialized and configured expands one's troubleshooting skills dramatically. The process is detailed below:

- A wireless card is inserted in a PCMCIA slot.
- The card services manager, which is listening in the background, detects the event, queries the card as to its function, brand, chipset, etc. The card manager takes this information and evaluates it. If the information is deemed to be correct (that is, without logical error), the card manager takes this information and moves to the next step. If the card manager recieves information that it believes is not correct, or in error to what its programmed to expect, an error is displayed and the card manager goes back to sleep.
- With the card's identity in hand, the CS manager goes to the */etc/pcmcia* directory and tries to match the information it has with the information contained in the various configuration files there. It starts with the *config* file, and scans it for a vendor/chipset match. If a match is found, the details of the match tell the card manager what the function of the device is, and where to go for further configuration information (e.g., from *config* to... *network* [device]? *scsi* [device]? *memory* [card]? etc.). If no exact match is found, the card manager (depending on the scenario) does one of two things: Gives up, or tries to resolve the discrepancy by guessing.
- As noted, the extension-less configuration files contains a set of default settings and the *.opts* file contains user-defined settings. So the card manager parses the first, and then applies any further settings it finds in the second. Using a wireless network card as an example, the full configuration process looks like this: *config* --> *config.opts* --> *network* --> *network.opts* --> *wireless* --> *wireless.opts*.

If all things have gone according to plan, your PCMCIA device should now be functioning. The next panel details some of the reasons why your device might not be working.

When things go wrong

If your device is not working as advertised, you should now have a better understanding of why. Here are some possible reasons:

- The card services manager could not match the information the device provided with what it found in */etc/pcmcia/config* (you insert your card, it beeps, and nothing happens after that).
- It "guessed" wrong (you insert your card, it beeps, all the LEDs light up, but nothing works as advertised).
- You entered the wrong information in the *opts* configuration file (similar scenario to the previous one; the card looks like it's working, but doesn't work as advertised)

Configuring *config.opts*

In most scenarios, the only two files you'll have to edit to get a wireless card up and running are *network.opts* and, on occasion, *wireless.opts* (usually putting the correct network parameters in *network.opts* will suffice). Some notebook brands, however, do not play well with certain network cards. The reason is the notebook (or a device loaded by your operating system) and the network card are both hard-coded to use the same memory register. The solution for this lies in the *config.opts* file.

```
# System resources available for PCMCIA devices

include port 0x100-0x4ff, port 0xc00-0xcff
include memory 0xc0000-0xfffff
include memory 0xa0000000-0xa0ffffff, memory 0x60000000-0x60ffffff
```

The *include* option statements tell the PCMCIA device what memory/port options are available for use. If inserting a card causes your system to lock or behave in erratic ways, go to the Linux on Laptops site (see Resources on page 20) and dig through the documentation there specific to your machine. For example, to configure wireless networking on a Dell Inspiron 8000 running SuSE 7.1, you must add **`port 0x800-0x8ff`** to the `include port...` line shown above.

If you're feeling brave and don't mind experimenting at the cost of possibly locking up your system to a hard reset, you can free up some system resources and possibly increase device performance by tinkering with the settings listed in this file.

Configuring *network.opts*

The next file up for discussion is */etc/pcmcia/network.opts*. As the name implies, *network.opts* contains your system's network configuration settings. The format of the file is basically self-explanatory. If your network is configured with DHCP, simply adjust the `DHCP` lines shown below accordingly.

```
    # Use DHCP (via /sbin/dhcpcd, /sbin/dhclient, or /sbin/pump)? [y/n]
DHCP="n"
```

```
# If you need to explicitly specify a hostname for DHCP requests
DHCP_HOSTNAME="localhost"
```

If your system uses a static IP, the pertinent lines are:

```
    # Host's IP address, netmask, network address, broadcast address
IPADDR="192.168.1.9"
NETMASK="255.255.255.0"
NETWORK="192.168.1.0"
BROADCAST="192.168.1.255"
```

Last but not least, enter the address of your network router (if you're running DHCP, you might not need an entry here).

```
    # Gateway address for static routing
GATEWAY="192.168.1.2"
```

Optional *config.opts* settings

The balance of the settings contained in *config.opts* are optional and are provided for users with unique needs. For example, you might use a built-in 10/100 connection at work and a wireless connection at home. In such a scenario, you don't want to use the settings found in */etc/resolv.conf* for wireless connectivity. The `DOMAIN`, `SEARCH`, and `DNS_*` entries allow you to specify one set of settings for the 10/100 interface and another for your wireless card.

```
    # Things to add to /etc/resolv.conf for this interface
DOMAIN=""
SEARCH=""
DNS_1=""
DNS_2=""
DNS_3=""
# NFS mounts, should be listed in /etc/fstab
MOUNTS=""
# If you need to override the interface's MTU...
MTU=""
# For IPX interfaces, the frame type and network number
IPX_FRAME=""
IPX_NETNUM=""
# Extra stuff to do after setting up the interface
start_fn () { return; }
# Extra stuff to do before shutting down the interface
stop_fn () { return; }
# Card eject policy options
NO_CHECK=n
NO_FUSER=n
```

TIP: When first configuring a new device, add only what you need to get it to a working state. Once you're satisfied with the basics, *then* go back a add in any optional parameters. It's a lot easier to troubleshoot a configuration that contains only three or four entries than it is one with 10 or 12.

Configuring *wireless.opts*

The last file you might need to edit is */etc/pcmcia/wireless.opts*. The primary purpose for this configuration file is to help your wireless card find the correct access point. In most scenarios there's only one available, so the following script block should do the trick:

```
# Here is an example of scheme matching
# Activate with "cardctl scheme essidany"

# Pick up any Access Point, should work on most 802.11 cards
essidany,*,*,*)
    INFO="Any ESSID"
    ESSID="linksys"
    ;;
```

Change the `ESSID` entry (station identifier; similar to a hostname in function) to match your WAP, and you should be all set. Some WAPs have different requirements; read through the supplied examples or consult the documentation that came with your device if you experience any difficulties.

Sidebar: Securing your wireless connection

Recent developments in technology have, almost overnight, brought wireless networking to the general computing public. For the computer enthusiast, wireless is clearly "the way of the future." Almost all new electronic devices sold on the market today have some kind of provision for wireless connectivity either built in or available as some form of add-on widget. This trend, however, has brought with it a growing concern for the security issues inherent in wireless networking.

On a wired network, information is exchanged between systems -- by default -- as a flow of plain text datagrams easily read by anyone with a basic knowledge of packet sniffing. Wireless networks increase the potential vulnerability of your data because their intrinsic "seamless" nature means hackers no longer have to concern themselves with the logistics of gaining access to the network itself; they can simply capture and read packets floating around in the ether. The solution too is simple: Encrypt the data passed between wireless devices.

The process of encrypting wireless tranmissions is a relatively simple one; however, the actual implementation is device-specific. Securing your wireless network can be accomplished by completing the following general steps:

- On your wireless access point, enable WEP (Wireless Encryption Protocol).
- Choose an encryption strength (most devices provide a range of "bit-strengths"). The higher the encryption strength, the more secure your data. On the other hand, the higher the strength, the higher the resulting latency because each device must encrypt and decrypt the datagrams exchanged. As a rule, 128-bit is considered more than adequate.
- Supply a passphrase (the longer the better) and instruct your WAP to generate a key (or keys).
- Cut and paste the resultant key into a plain text editor for the moment.
- Open the appropriate configuration file on the system(s) you will use to communicate with the WAP. Enable encryption, and paste the key generated by the WAP into file. The next panel contains details on encryption configuration.

For further details on configuring your WAP for wireless encryption, please see the documentation supplied with the product.

Securing your wireless connection

As noted in the sidebar in the previous panel, if you decide to encrypt your wireless connection you'll need to implement the discussed changes to your access point *and* provide your wireless card with the necessary keys so it can encrypt/decrypt the packets received by your system. This key information is added to */etc/pcmcia/wireless.opts*. The following example shows one of the more common formats for this information:

```
KEY="4567-89AB-CD, s:passphrase"
```

The first part of the entry is the key (as generated by the access point), and the second is the passphrase entered to generate the key. Unfortunately, every vendor has its own little quirk as to how they want this information presented, and if you get it wrong, you won't be able to connect to your WAP. Again, consult the documentation supplied with both your WAP and your network card, and read through the examples provided in *wireless.opts*.

Section 6. Summary and further resources

Summary

This tutorial detailed one method of configuring wireless networking under Linux. The importance of knowing your card's underlying chipset, and locating the correct driver for that chipset was stressed. Addressing these fundamentals goes a long way toward removing the frustration and "head scratching" some users experience when confronted with the ins-and-outs of establishing wireless connectivity.

The key points discussed in this document were:

- Disabling the Linux kernel services and recompiling the kernel
- Obtaining and installing the PCMCIA-CS source
- Locating and installing a 3rd.-party device driver for your wireless card (in situations that warrant it)
- Configuring your wireless device

Due to the wide range of devices on the market, wireless connectivity can be a thorny beast at times. The following panels provide a further source of resources pertaining to the topic of wireless networking. If, after completing this tutorial, you still have issues or questions, feel free to contact the author for more information.

Resources

For further information pertaining to basic system administration, wireless networking HOWTOs, configuration issues, and downloads, please see one or more of the following resources.

For additional material on system administration:

- *Practical UNIX & Internet Security* (Garfinkel and Spafford, *O'Reilly & Associates*) is a must-have bookshelf reference for anyone tasked with maintaining and securing *NIX systems. An excellent companion to this text is O'Reilly's *UNIX Power Tools* (Peek, O'Reilly, and Loukides,) which contains thousands of useful tips, scripts, and insights relating to the daily management of *NIX environments.
- *Special Edition: Using Linux* (Bandel and Napier, *Que Publishing*) is a good comprehensive introduction to using, maintaining, and administering a wide variety of Linux distributions.
- *Linux in a Nutshell* (Siever, Spainhour, Figgins, and Hekman) and *UNIX in a Nutshell* (Robbins), both published by O'Reilly & Associates, are the *sine qua non* of functional, quick reference guides to the sometime cryptic command-line syntax associated with Linux/UNIX operating systems. Highly recommended.
- One of the best known, and most frequently used, sources for online Linux documentation is *The Linux Documentation Project*. Here you'll find an extensive compilation of HOWTOs on a wide variety of topics pertaining to the configuration, customization, maintenance, and administration of Linux. Be sure to bookmark this one.
- Another good resource for administrative material is *The Linux System Administrator's Guide*; it can be found online at *www.tldp.org/LDP/sag/*. Topics covered include filesystem

layout, device management, memory management, user management, program installation, a good explanation of the Linux initialization process, and a brief overview of common system services.
- IBM has a Redbook available entitled, *Linux System Administration and Backup Tools for IBM xSeries and Netfinity*. If you put aside the IBM-specific hardware notes, there's some good generic system administration tips and techniques in this publication.

More resources

There's a vast amount of online material pertaining to wireless networking and configuration available on the Web, and the following links should get you headed in the right direction. If you can't find what you're looking for in any of the resources listed, try searching *Google* using device- or task-specific keywords.

For further information on PCMCIA and wireless-related topics, try the following sites:

- The *Linux PCMCIA Information Page* is the place to go for all things PCMCIA-CS-related: Documentation, download links, release notes, lists of supported cards, device driver links and resources, and message forum/mailing list information.
- For access to Linux kernel source code, go to the *Linux Kernel Archives* Web site (or a local mirror).
- Jean Tourrilhes maintains an extensive page of Linux wireless device information subtitled *The who's who of Wireless LANs under Linux*. If you're looking for a comprehensive list of common wireless devices compatible with Linux, along with details on their underlying chipsets, this is it. Not only does Jean's page contain lots of good product information, there's also links for vendor sites, mailing lists, 3rd.-party drivers, etc.
- *Linksys* is a popular vendor of good quality, reasonably priced wireless products, which I've personally used for several years. Their tech support staff are knowledgable and prompt in replying to queries.
- If your wireless device happens to be compatible with the *wlan* or *wlan-ng* device driver, check out *The Linux wlan(tm) Project* hosted and supported by AbsoluteValue Systems, Inc. There's lot of good documenation and FAQs to be found here, as well as source tarballs of the *wlan* drivers. If you're having problems installing or configuring any aspect of the *wlan* driver, check out one of the *available mailing lists*.
- If your wireless card is based on the Intersil's Prism2/2.5/3 chipset, you'll find drivers, CVS snapshots, changelogs, documentation, and FAQs available from *this site*.
- See the *Linux on Laptops* site for information and resources for running Linux on a laptop.

Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial

generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.