Linux PCMCIA HOWTO

Table of Contents

Table of Contents

Table of Contents

Linux PCMCIA HOWTO

David Hinds, `dahinds@users.sourceforge.net`.

v2.77, 18 September 2000

This document describes how to install and use PCMCIA Card Services for Linux, and answers some frequently asked questions. The latest version of this document can always be found at ftp://projects.sourceforge.net/pub/pcmcia−cs/doc. An HTML version is at http://pcmcia−cs.sourceforge.net.

1.General information and hardware requirements

2.Compilation and installation

3.Resolving installation and configuration problems

4.Usage and features

5.Advanced topics

6.Dealing with unsupported cards

7.Debugging tips and programming information

1.General information and hardware requirements

1.1 Introduction

Card Services for Linux is a complete PCMCIA or ``PC Card'' support package. It includes a set of loadable kernel modules that implement a version of the Card Services applications program interface, a set of client drivers for specific cards, and a card manager daemon that can respond to card insertion and removal events, loading and unloading drivers on demand. It supports ``hot swapping'' of most card types, so cards can be safely inserted and ejected at any time.

This software is continually under development. It probably contains bugs, and should be used with caution. I'll do my best to fix problems that are reported to me, but if you don't tell me, I may never know. If you use this code, I hope you will send me your experiences, good or bad!

If you have any suggestions for how this document could be improved, please let me know (`dahinds@users.sourceforge.net`).

1.2 Copyright notice and disclaimer

1.3 What is the latest version, and where can I get it?

The current major release of Card Services is version 3.1, and minor updates or bug fixes are numbered 3.1.1, 3.1.2, and so on.

Source code for the latest version is available by FTP at `projects.sourceforge.net` in the `/pub/pcmcia-cs` directory, as `pcmcia-cs-3.1.?.tar.gz`. There will usually be several versions here. I generally only keep the latest minor release for a given major release. New major releases may contain relatively untested code, so I also keep the latest version of the previous major release as a relatively stable fallback; the current fallback is 3.0.14. It is up to you to decide which version is more appropriate, but the `CHANGES` file will summarize the most important differences.

The Linux PCMCIA FTP site is mirrored at `sunsite.unc.edu` (and all sunsite mirror sites) in `/pub/Linux/kernel/pcmcia`.

If you do not feel up to compiling the drivers from scratch, pre−compiled drivers are included with current releases of most of the major Linux distributions, including Slackware, Debian, Red Hat, Caldera, SuSE, and Yggdrasil, among others.

1.4 What systems are supported?

This package should run on almost Intel−based Linux−capable laptop. It also runs on Alpha−based platforms (i.e., the DEC Multia). Work is being done to make the package fully dual−endian, so that it will also support PowerPC−based platforms (i.e., Apple Powerbooks). Most common socket controllers are supported. Card docks for desktop systems should work as long as they use a supported controller, and are plugged directly into the ISA or PCI bus, as opposed to SCSI−to−PCMCIA or IDE−to−PCMCIA adapters. The following controllers are recognized by the supplied socket drivers:

- Cirrus Logic PD6710, PD6720, PD6722, PD6729, PD6730, PD6732, PD6832
- Intel i82365sl B, C, and DF steps, 82092AA
- O2Micro OZ6729, OZ6730, OZ6812, OZ6832, OZ6833, OZ6836, OZ6860
- Omega Micro 82C365G, 82C092G
- Ricoh RF5C296, RF5C396, RL5C465, RL5C466, RL5C475, RL5C476, RL5C478
- SMC 34C90
- Texas Instruments PCI1031, PCI1130, PCI1131, PCI1210, PCI1211, PCI1220, PCI1221, PCI1225, PCI1250A, PCI1251A, PCI1251B, PCI1410, PCI1420, PCI1450
- Toshiba ToPIC95, ToPIC97, ToPIC100 (experimental, incomplete)
- Vadem VG465, VG468, VG469
- VLSI Technologies 82C146, VCF94365
- VIA VT83C469
- Databook DB86082, DB86082A, DB86084, DB86084A, DB86072, DB86082B

Other controllers that are register compatible with the Intel i82365sl will generally work, as well.

Support for 32−bit CardBus cards is still somewhat experimental. Drivers prior to version 3.0 only support 16−bit cards in CardBus sockets. Due to the rapid pace of technological change for laptop hardware, new controllers appear frequently, and there may be delays between when a new model appears on the market, and when driver support becomes available.

Toshiba has made available some documentation for their ToPIC95 and ToPIC97 chipsets, however the information they have provided has not really been adequate. Despite conflicting reports to the contrary, Toshiba has not made any effective effort to remedy this situation. There are serious bugs in Linux support for the ToPIC chipsets, that cannot be resolved until better documentation or help from Toshiba becomes available. I do not recommend use of Toshiba laptops at this time. For use of 16−bit cards, I recommend setting the bridge mode to ``PCIC'' in the BIOS setup; for CardBus cards, you are on your own.

The Motorola 6AHC05GA controller used in some Hyundai laptops is not supported. The custom host controller in the HP Omnibook 600 is also unsupported.

1.5 What cards are supported?

The current release includes drivers for a variety of ethernet cards, a driver for modem and serial port cards, several SCSI adapter drivers, a driver for ATA/IDE drive cards, and memory card drivers that should support most SRAM cards and some flash cards. The `SUPPORTED.CARDS` file included with each release of Card Services lists all cards that are known to work in at least one actual system.

The likelihood that a card not on the supported list will work depends on the type of card. Essentially all modems should work with the supplied driver. Some network cards may work if they are OEM versions of supported cards. Other types of IO cards (frame buffers, sound cards, etc) will not work until someone writes the appropriate drivers.

1.6 When will my favorite (unsupported) card become supported?

Unfortunately, they usually don't pay me to write device drivers, so if you would like to have a driver for your favorite card, you are probably going to have to do at least some of the work. Ideally, I'd like to work towards a model like the Linux kernel, where I would be responsible mainly for the ``core'' driver code and other authors would contribute and maintain client drivers for specific cards. The `SUPPORTED.CARDS` file mentions some cards for which driver work is currently in progress. I will try to help where I can, but be warned that debugging kernel device drivers by email is not particularly effective.

Manufacturers interested in helping provide Linux support for their products can contact me about consulting arrangements.

1.7 Mailing lists and other information sources

I used to maintain a database and mailing list of Linux PCMCIA users. More recently, I've turned my web page for Linux PCMCIA information into a ``HyperNews'' site, with a set of message lists for Linux PCMCIA issues. There are lists for installation and configuration issues, for different types of cards, and for programming and debugging issues. The Linux PCMCIA information page is at http://pcmcia−cs.sourceforge.net. Users can request email notification of new responses to particular questions, or notification for all new messages in a given category. I hope that this will become a useful repository of information, for questions that go beyond the scope of the HOWTO.

There is a Linux mailing list devoted to laptop issues, the ``linux−laptop'' list. For more information, send a message containing the word ``help'' to `majordomo@vger.rutgers.edu`. To subscribe, send a message containing ``subscribe linux−laptop'' to the same address. This mailing list might be a good forum for discussion of Linux PCMCIA issues.

The Linux Laptop Home Page at http://www.cs.utexas.edu/users/kharker/linux−laptop has links to many sites that have information about configuring specific types of laptops for Linux. There is also a searchable database of system configuration information.

1.8 Why don't you distribute binaries?

For me, distributing binaries would be a significant hassle. It is complicated because some features can only be selected at compile time, and because the modules are somewhat dependent on having the ``right'' kernel configuration. So, I would probably need to distribute precompiled modules along with matching kernels. Beyond this, the greatest need for precompiled modules is when installing Linux on a clean system. This typically requires setting up drivers so they can be used in the installation process for a particular Linux distribution. Each Linux distribution has its own ideosyncracies, and it is not feasible for me to provide boot and root disks for even just the common combinations of drivers and distributions.

PCMCIA is now a part of many of the major Linux distributions, including Red Hat, Caldera, Slackware, Yggdrasil, Craftworks, and Nascent Technology.

1.9 Why is the package so darned big?

Well, first of all, it isn't actually that large. All the driver modules together take up about 500K of disk space. The utility programs add up to about 70K, and the scripts in `/etc/pcmcia` are about 50K. The core driver modules take up about 55K of system memory. The `cardmgr` daemon will generally be swapped out except when cards are inserted or removed. The total package size is comparable to DOS/Windows Card Services implementations.

Compared to DOS ``point enablers'', this may still seem like a lot of overhead, especially for people that don't plan on using many of the features of PCMCIA, such as power management or hot swapping. Point enablers can be tiny because they generally support only one or a small set of cards, and also generally support a restricted set of host controllers. If someone were to write a genuinely ``generic'' modem enabler, it would end up incorporating much of the functionality of Card Services, to handle cards from different vendors and the full range of host controller variants.

2.Compilation and installation

2.1 Prerequisites and kernel setup

Before starting, you should think about whether you really need to compile the PCMCIA package yourself. All common Linux distributions come with pre−compiled driver packages. Generally, you only need to install the drivers from scratch if you need a new feature of the current drivers, or if you've updated and/or reconfigured your kernel in a way that is incompatible with the drivers included with your Linux distribution. While compiling the package is not technically difficult, it does require some general Linux familiarity.

The following things should be installed on your system before you begin:

- A 2.0, 2.2, 2.3, or 2.4 series kernel source tree.
- An appropriate set of module utilities.

* (Optional) the ``XForms'' X11 user interface toolkit.

You need to have a complete linux source tree for your kernel, not just an up−to−date kernel image. The driver modules contain some references to kernel source files. While you may want to build a new kernel to remove unnecessary drivers, installing PCMCIA does not require you to do so.

Current ``stable'' kernel sources and patches are available from ftp://ftp.us.kernel.org/pub/linux/kernel/v2.2. Development kernels can be found in the corresponding `v2.3` or `v2.4` subdirectories. Current module utilities can be found in the same locations.

In the Linux kernel source tree, the `Documentation/Changes` file describes the versions of all sorts of other system components that are required for that kernel release. You may want to check through this and verify that your system is up to date, especially if you have updated your kernel. If you are using a development kernel, be sure that you are using the right combination of shared libraries and module tools.

When configuring your kernel, if you plan on using a PCMCIA ethernet card, you should turn on networking support but turn off the normal Linux network card drivers, including the ``pocket and portable adapters''. The PCMCIA network card drivers are all implemented as loadable modules. Any drivers compiled into your kernel will only waste space.

If you want to use SLIP, PPP, or PLIP, you do need to either configure your kernel with these enabled, or use the loadable module versions of these drivers. There is an unfortunate deficiency in the kernel config process in 1.2.X kernels, in that it is not possible to set configuration options (like SLIP compression) for a loadable module, so it is probably better to just link SLIP into the kernel if you need it.

In order to use a PCMCIA token ring adapter, your kernel should be configured with ``Token Ring driver support'' (`CONFIG_TR`) enabled, though you should leave `CONFIG_IBMTR` off.

If you want to use a PCMCIA IDE adapter, your kernel should be configured with `CONFIG_BLK_DEV_IDE_PCMCIA` enabled, for 2.0.* through 2.1.7 kernels. Older kernels do not support removeable IDE devices; newer kernels do not require a special configuration setting.

If you will be using a PCMCIA SCSI adapter, then enable `CONFIG_SCSI` when configuring your kernel. Also, enable any top level drivers (SCSI disk, tape, cdrom, generic) that you expect to use. All low−level drivers for particular host adapters should be disabled, as they will just take up space.

If you want to modularize a driver that is needed for a PCMCIA device, you must modify `/etc/pcmcia/config` to specify what modules need to be loaded for what card types. For example, if the serial driver is modularized, then the serial device definition should be:

```
device "serial_cs"
  class "serial" module "misc/serial", "serial_cs"
```

This package includes an X−based card status utility called `cardinfo`. This utility is based on a freely distributed user interface toolkit called the XForms Library. This library is available as a separate package with most Linux distributions. If you would like to build `cardinfo`, you should install XForms and all the normal X header files and libraries before configuring the PCMCIA package.

2.2 Installation

Here is a synopsis of the installation process:

- Unpack pcmcia–cs–3.1.?.tar.gz in /usr/src.
- Run ``make config'' in the new `pcmcia-cs-3.1.?` directory.
- Run ``make all'', then ``make install''.
- Customize the startup script and the option files in `/etc/pcmcia` for your site, if needed.

If you plan to install any contributed client drivers not included in the core PCMCIA distribution, unpack each of them in the top–level directory of the PCMCIA source tree. Then follow the normal build instructions. The extra drivers will be compiled and installed automatically.

Running ``make config'' prompts for a few configuration options, and checks out your system to verify that it satisfies all prerequisites for installing PCMCIA support. In most cases, you'll be able to just accept all the default configuration options. Be sure to carefully check the output of this command in case there are problems. The following options are available:

Alternate target install directory?

If you are compiling the package for installation on another machine, specify an alternate target directory when prompted. This should be an absolute path. All files will be installed relative to this directory. You will then be able to `tar` this directory tree and copy to your target machine, and unpack relative to its root directory to install everything in the proper places. Newer PCMCIA releases do not ask for this; instead it can be set with the `--target=` command line option to the Configure script.

Build 'trusting' versions of card utilities?

Some of the support utilities (`cardctl` and `cardinfo`) can be compiled either in ``safe'' or ``trusting'' forms. The ``safe'' forms prevent non–root users from modifying card configurations. The ``trusting'' forms permit ordinary users to issue commands to suspend and resume cards, reset cards, and change the current configuration scheme. The default is to build the safe forms.

Include 32–bit (CardBus) card support?

This option must be selected if you wish to use 32–bit CardBus cards. It is not required for CardBus bridge support, if you only plan to use 16–bit PC Cards.

Include PnP BIOS resource checking?

This builds additional code into the PCMCIA core module to communicate with a system's PnP BIOS to obtain resource information for built–in ``motherboard'' devices (serial and parallel ports, sound, etc), to help avoid resource conflicts. If enabled, some extra resource files will be created under `/proc/bus/pccard`, and the `lspnp` and `setpnp` tools can be used to view and manipulate PnP BIOS devices. However, this setting causes problems on some laptops and is not turned on by default.

Module install directory?

The directory that new kernel modules will be installed into. Normally this should be the subdirectory of `/lib/modules` that matches your kernel version.

How to set kernel−specific options?

There are a few kernel configuration options that affect the PCMCIA tools. The configuration script can deduce these from the running kernel (the default and most common case). Alternatively, if you are compiling for installation on another machine, it can read the configuration from a kernel source tree, or each option can be set interactively.

The `Configure` script can also be executed non−interactively, for automatic builds or to quickly reconfigure after a kernel update. Some additional less−frequently−used options can be only be set from the command line. Running ``Configure --help'' lists all available options.

Running ``make all'' followed by ``make install'' will build and then install the kernel modules and utility programs. Kernel modules are installed under `/lib/modules/<version>/pcmcia`. The `cardmgr` and `cardctl` programs are installed in `/sbin`. If `cardinfo` is built, it is installed in `/usr/bin/X11`.

Configuration files will be installed in the `/etc/pcmcia` directory. If you are installing over an older version, your old config scripts will be backed up before being replaced. The saved scripts will be given an `*.O` extension.

If you don't know what kind of host controller your system uses, you can use the `probe` utility in the `cardmgr/` subdirectory to determine this. There are two major types: the Databook TCIC−2 type and the Intel i82365SL−compatible type.

In a few cases, the `probe` command will be unable to determine your controller type automatically. If you have a Halikan NBD 486 system, it has a TCIC−2 controller at an unusual location: you'll need to edit `rc.pcmcia` to load the `tcic` module, and also set the `PCIC_OPTS` parameter to ``tcic_base=0x02c0''.

On some systems using Cirrus controllers, including the NEC Versa M, the BIOS puts the controller in a special suspended state at system startup time. On these systems, the `probe` command will fail to find any known host controller. If this happens, edit `rc.pcmcia` and set `PCIC` to `i82365`, and `PCIC_OPTS` to ``wakeup=1''.

2.3 Startup options

The PCMCIA startup script recognizes several groups of startup options, set via environment variables. Multiple options should be separated by spaces and enclosed in quotes. Placement of startup options depends on the Linux distribution used. They may be placed directly in the startup script, or they may be kept in a separate option file. See the Notes about specific Linux distributions for specifics. The following variables can be set:

`PCMCIA`

This variable specifies whether PCMCIA support should be started up, or not. If it is set to anything other than ``yes'', then the startup script will be disabled.

PCIC

This identifies the PC Card Interface Controller driver module. There are two options: ``tcic'' or ``i82365''. Virtually all current controllers are in the ``i82365'' group. This is the only mandatory option setting.

PCIC_OPTS

This specifies options for the PCIC module. Some host controllers have optional features that may or may not be implemented in a particular system. In some cases, it is impossible for the socket driver to detect if these features are implemented. See the corresponding man page for a complete description of the available options.

CORE_OPTS

This specifies options for the `pcmcia_core` module, which implements the core PC Card driver services. See ``man pcmcia_core'' for more information.

CARDMGR_OPTS

This specifies options to be passed to the `cardmgr` daemon. See ``man cardmgr'' for more information.

SCHEME

If set, then the PC Card configuration scheme will be initialized to this at driver startup time. See the Overview of the PCMCIA configuration scripts for a discussion of schemes.

The low level socket drivers, `tcic` and `i82365`, have various bus timing parameters that may need to be adjusted for certain systems with unusual bus clocking. Symptoms of timing problems can include card recognition problems, lock−ups under heavy loads, high error rates, or poor device performance. Only certain host bridges have adjustable timing parameters: check the corresponding man page to see what options are available for your controller. Here is a brief summary:

- Cirrus controllers have numerous configurable timing parameters. The most important seems to be the `cmd_time` flag, which determines the length of PCMCIA bus cycles. Fast 486 systems (i.e., DX4−100) seem to often benefit from increasing this from 6 (the default) to 12 or 16.
- The Cirrus PD6729 PCI controller has the `fast_pci` flag, which should be set if the PCI bus speed is greater than 25 MHz.
- For Vadem VG−468 controllers, the `async_clock` flag changes the relative clocking of PCMCIA bus and host bus cycles. Setting this flag adds extra wait states to some operations. However, I have yet to hear of a laptop that needs this.
- The `pcmcia_core` module has the `cis_speed` parameter for changing the memory speed used for accessing a card's Card Information Structure (CIS). On some systems with fast bus clocks, increasing this parameter (i.e., slowing down card accesses) may be beneficial for card recognition problems.
- This is not a timing issue, but if you have more than one ISA−to−PCMCIA controller in your system

or extra sockets in a laptop docking station, the `i82365` module should be loaded with the `extra_sockets` parameter set to 1. This should not be necessary for detection of PCI−to−PCMCIA or PCI−to−CardBus bridges.

Here are some timing settings for specific systems:

* On the ARM Pentium−90 or Midwest Micro Soundbook Plus, use ``freq_bypass=1 cmd_time=8''.
* On a Midwest Micro Soundbook Elite, use ``cmd_time=12''.
* On a Gateway Liberty, try ``cmd_time=16''.
* On a Samsung SENS 810, use ``fast_pci=1''.

Card readers for desktop systems

While almost all PCMCIA card readers and card docks work fine under Linux, some require special startup options because they do not behave exactly like laptop PCMCIA bridges. PCI card readers, in particular, handle interrupts differently.

* The Linksys ProConnect PCMRDWR and Antec DataChute card readers are ``ISA Plug and Play'' devices. To use these, you must first activate them with the Linux isapnp tools. See the man pages for `pnpdump` and `isapnp` for more information.
* For Elan P−series PCI card readers based on the Cirrus PD6729 PCI−to−PCMCIA bridge chip, the `i82365` driver requires a ``irq_mode=1'' parameter.
* For the Sycard PCChost1200 host adapter, the `i82365` driver requires a ``p2cclk=1'' parameter.
* With SCM Microsystems SBP series PCI card readers (which are also being distributed with Lucent WaveLAN IEEE cards), it is necessary to specify ``irq_mode=0'' for the `i82365` module, to force use of PCI interrupts.
* For the ActionTec PC 750 card reader, the `i82365` driver requires a ``irq_list=0'' parameter, to indicate that ISA interrupts are unavailable.

2.4 System resource settings

Card Services should automatically avoid allocating IO ports and interrupts already in use by other standard devices. It will also attempt to detect conflicts with unknown devices, but this is not completely reliable. In some cases, you may need to explicitly exclude resources for a device in `/etc/pcmcia/config.opts`.

Here are some resource settings for specific laptop types. View this list with suspicion: it may give useful hints for solving problems, but it is inevitably out of date and certainly contains mistakes. Corrections and additions are welcome.

* On the AMS SoundPro, exclude irq 10.
* On some AMS TravelPro 5300 models, use memory 0xc8000−0xcffff.
* On the BMX 486DX2−66, exclude irq 5, irq 9.
* On the Chicony NB5, use memory 0xda000−0xdffff.

* On the Compaq Presario 1020, exclude port 0x2f8−0x2ff, irq 3, irq 5.
* On the Dell Inspiron 7000, exclude irq 3, irq 5.
* On the Fujitsu C series, exclude port 0x200−0x27f.
* On the HP Omnibook 4000C, exclude port 0x300−0x30f.
* On the IBM ThinkPad 380, and maybe the 385 and 600 series, exclude port 0x230−0x233, and irq 5.
* On IBM ThinkPad 600 and 770 models with internal modems, exclude port 0x2f8−0x2ff.
* On the IBM ThinkPad 600E and 770Z, change the high memory window to 0x60000000−0x60ffffff.
* On the Micron Millenia Transport, exclude irq 5, irq 9.
* On the NEC Versa M, exclude irq 9, port 0x2e0−2ff.
* On the NEC Versa P/75, exclude irq 5, irq 9.
* On the NEC Versa S, exclude irq 9, irq 12.
* On the NEC Versa 6000 series, exclude port 0x2f8−0x33f, irq 9, irq 10.
* On the NEC Versa SX, exclude port 0x300−0x31f.
* On the ProStar 9200, Altima Virage, and Acquiline Hurricane DX4−100, exclude irq 5, port 0x330−0x35f. Maybe use memory 0xd8000−0xdffff.
* On the Siemens Nixdorf SIMATIC PG 720C, use memory 0xc0000−0xcffff, port 0x300−0x3bf.
* On the TI TravelMate 5000, use memory 0xd4000−0xdffff.
* On the Toshiba Satellite 4030CDS, exclude irq 9.
* On the Toshiba T4900 CT, exclude irq 5, port 0x2e0−0x2e8, port 0x330−0x338.
* On the Toshiba Tecra 8000, exclude irq 3, irq 5, irq 9.
* On the Twinhead 5100, HP 4000, Sharp PC−8700 and PC−8900, exclude irq 9 (sound), irq 12.
* On an MPC 800 Series, exclude irq 5, port 0x300−0x30f for the CD−ROM.

PowerBook specific settings

On PowerPC based PowerBook systems, the default system resources in `/etc/pcmcia/config.opts` file are no good at all. Replace all the IO port and window definitions with something like:

```
include port 0x100-0x4ff, port 0x1000-0x17ff
include memory 0x80000000-0x80ffffff
```

2.5 Notes about specific Linux distributions

This section is incomplete. Corrections and additions are welcome.

Debian

Debian uses a System V boot script arrangement. The PCMCIA startup script is installed as `/etc/init.d/pcmcia`, and startup options are specified in `/etc/pcmcia.conf`. Debian's syslog configuration will place kernel messages in `/var/log/messages` and `cardmgr` messages in `/var/log/daemon.log`.

Debian distributes the PCMCIA system in two packages: the ``pcmcia-cs'' package contains cardmgr and other tools, man pages, and configuration scripts; and the ``pcmcia-modules'' package contains the kernel driver modules.

Red Hat, Caldera, Mandrake

These distributions use a System V boot script organization. The PCMCIA startup script is installed as /etc/rc.d/init.d/pcmcia, and boot options are kept in /etc/sysconfig/pcmcia. Beware that installing the Red Hat package may install a default boot option file that has PCMCIA disabled. To enable PCMCIA, the ``PCMCIA'' variable should be set to ``yes''. Red Hat's default syslogd configuration will record all interesting messages in /var/log/messages.

Red Hat's PCMCIA package contains a replacement for the network setup script, /etc/pcmcia/network, which meshes with the Red Hat linuxconf configuration system. This is convenient for the case where just one network adapter is used, with one set of network parameters, but does not have the full flexibility of the regular PCMCIA network script. Compiling and installing a clean PCMCIA source distribution will overwrite the network script, breaking the link to the Red Hat tools. If you prefer using the Red Hat tools, either use only Red Hat RPM's, or replace /etc/pcmcia/network.opts with the following:

```
if [ -f /etc/sysconfig/network-scripts/ifcfg-$2 ] ; then
    start_fn () {
        . /etc/sysconfig/network-scripts/ifcfg-$1
        if [ "$ONBOOT" = "yes" ] ; then /sbin/ifup $1 ; fi
    }
    stop_fn () {
        /sbin/ifdown $1
    }
fi
```

If you do use linuxconf (or netconf) to configure your network interface, leave the ``kernel module'', ``I/O port'', and ``irq'' parameters blank. Setting these parameters may interfere with proper operation of the PCMCIA subsystem.

At boot time, when the Red Hat network subsystem starts up, it may say ``Delaying eth0 initialization'' and ``[FAILED]''. This is actually not a failure: it means that this network interface will not be initialized until after the PCMCIA network device is configured.

Red Hat bundles their slightly modified PCMCIA source distribution with their kernel sources, rather than as a separate source package. When preparing to build a new set of PCMCIA drivers, you will generally want to install Red Hat's kernel–source RPM (kernel-source-*.i386.rpm), and not the kernel SRPM (kernel-*.src.rpm). The SRPM is tailored for building their kernel RPM files, which is not exactly what you want.

Slackware

Slackware uses a BSD boot script arrangement. The PCMCIA startup script is installed as `/etc/rc.d/rc.pcmcia`, and boot options are specified in `rc.pcmcia` itself. The PCMCIA startup script is invoked from `/etc/rc.d/rc.S`.

SuSE

SuSE uses a System V init script arrangement, with init scripts stored under `/sbin/init.d`. The PCMCIA startup script is installed as `/sbin/init.d/pcmcia`, and startup options are kept in `/etc/rc.config`. The SuSE startup script is somewhat limited and does not allow PCMCIA startup variables to be overridden from the `lilo` boot prompt.

3.Resolving installation and configuration problems

This section describes some of the most common failure modes for the PCMCIA subsystem. Try to match your symptoms against the examples. This section only describes general failures that are not specific to a particular client driver or type of card.

Before trying to diagnose a problem, you have to know where your system log is kept (see Notes about specific Linux distributions). You should also be familiar with basic diagnostic tools like `dmesg` and `lsmod`. Also, be aware that most driver components (including all the kernel modules) have their own individual man pages.

In 3.1.15 and later releases, the `debug-tools` subdirectory of the PCMCIA source tree has a few scripts to help diagnose some of the most common configuration problems. The `test_setup` script checks your PCMCIA installation for completeness. The `test_network` and `test_modem` scripts will try to diagnose problems with PCMCIA network and modem cards. These scripts can be particularly helpful if you are unfamiliar with Linux and are not sure how to approach a problem.

Try to define your problem as narrowly as possible. If you have several cards, try each card in isolation, and in different combinations. Try cold Linux boots, versus warm boots from Windows. Compare booting with cards inserted, versus inserting cards after boot. If you normally use your laptop docked, try it undocked. And sometimes, two sockets will behave differently.

It is nearly impossible to debug driver problems encountered when attempting to install Linux via a PCMCIA device. Even if you can identify the problem based on its symptoms, installation disks are difficult to modify, especially without access to a running Linux system. Customization of installation disks is completely dependent on the choice of Linux distribution, and is beyond the scope of this document. In general, the best course of action is to install Linux using some other means, obtain the latest drivers, and then debug the problem if it persists.

3.1 Base PCMCIA kernel modules do not load

Symptoms:

- Kernel version mismatch errors are reported when the PCMCIA startup script runs.
- After startup, `lsmod` does not show any PCMCIA modules.
- `cardmgr` reports ``` ``no pcmcia driver in /proc/devices'' ``` in the system log.

Kernel modules contain version information that is checked against the current kernel when a module is loaded. The type of checking depends on the setting of the `CONFIG_MODVERSIONS` kernel option. If this is false, then the kernel version number is compiled into each module, and `insmod` checks this for a match with the running kernel. If `CONFIG_MODVERSIONS` is true, then each symbol exported by the kernel is given a sort of checksum. These codes are all compared against the corresponding codes compiled into a module. The intent was for this to make modules less version−dependent, because the checksums would only change if a kernel interface changed, and would generally stay the same across minor kernel updates. In practice, the checksums have turned out to be even more restrictive, because many kernel interfaces depend on compile−time kernel option settings. Also, the checksums turned out to be an excessively pessimistic judge of compatibility.

The practical upshot of this is that kernel modules are closely tied to both the kernel version, and the setting of many kernel configuration options. Generally, a set of modules compiled for one 2.0.31 kernel will not load against some other 2.0.31 kernel unless special care is taken to ensure that the two were built with similar configurations. This makes distribution of precompiled kernel modules a tricky business.

You have several options:

- If you obtained precompiled drivers as part of a Linux distribution, verify that you are using an unmodified kernel as supplied with that distribution. If you intend to use precompiled modules, you generally must stick with the corresponding kernel.
- If you have reconfigured or upgraded your kernel, you will probably need to compile and install the PCMCIA package from scratch. This is easily done if you already have the kernel source tree installed. See Compilation and installation for detailed instructions.
- In some cases, incompatibilities in other system components can prevent correct loading of kernel modules. If you have upgraded your own kernel, pay attention to the ``minimal requirements'' for module utilities and binutils listed in the `Documentation/Changes` file in the kernel source code tree.

3.2 Some client driver modules do not load

Symptoms:

- The base modules (`pcmcia_core`, `ds`, `i82365`) load correctly.
- Inserting a card gives a high beep + low beep pattern.
- `cardmgr` reports version mismatch errors in the system log.

Some of the driver modules require kernel services that may or may not be present, depending on kernel configuration. For instance, the SCSI card drivers require that the kernel be configured with SCSI support, and the network drivers require a networking kernel. If a kernel lacks a necessary feature, `insmod` may report undefined symbols and refuse to load a particular module. Note that `insmod` error messages do not distinguish between version mismatch errors and missing symbol errors.

Specifically:

- The serial client driver `serial_cs` requires the kernel serial driver to be enabled with `CONFIG_SERIAL`. This driver may be built as a module.
- Support for multiport serial cards or multifunction cards that include serial or modem devices requires `CONFIG_SERIAL_SHARE_IRQ` to be enabled.
- The SCSI client drivers require that `CONFIG_SCSI` be enabled, along with the appropriate top level driver options (`CONFIG_BLK_DEV_SD`, `CONFIG_BLK_DEV_SR`, etc for 2.1 kernels). These may be built as modules.
- The network client drivers require that `CONFIG_INET` is enabled. Kernel networking support cannot be compiled as a module.
- The token−ring client requires that the kernel be compiled with `CONFIG_TR` enabled.

There are two ways to proceed:

- Rebuild your kernel with the necessary features enabled.
- If the features have been compiled as modules, then modify `/etc/pcmcia/config` to preload these modules.

The `/etc/pcmcia/config` file can specify that additional modules need to be loaded for a particular client. For example, for the serial driver, one would use:

```
device "serial_cs"
  class "serial" module "misc/serial", "serial_cs"
```

Module paths are specified relative to the top−level module directory for the current kernel version; if no relative path is given, then the path defaults to the `pcmcia` subdirectory.

3.3 Interrupt scan failures

Symptoms:

- The system locks up when the PCMCIA drivers are loaded, even with no cards present.
- The system log shows a successful host controller probe just before the lock−up, but does not show interrupt probe results.

After identifying the host controller type, the socket driver probes for free interrupts. The probe involves programming the controller for each apparently free interrupt, then generating a ``soft'' interrupt, to see if the interrupt can be detected correctly. In some cases, probing a particular interrupt can interfere with another system device.

The reason for the probe is to identify interrupts which appear to be free (i.e., are not reserved by any other Linux device driver), yet are either not physically wired to the host controller, or are connected to another device that does not have a driver.

In the system log, a successful probe might look like:

```
Intel PCIC probe:
  TI 1130 CardBus at mem 0x10211000, 2 sockets
  ...
  ISA irqs (scanned) = 5,7,9,10 status change on irq 10
```

There are two ways to proceed:

- The interrupt probe can be restricted to a list of interrupts using the `irq_list` parameter for the socket drivers. For example, ``` ``irq_list=5,9,10" ``` would limit the scan to three interrupts. All PCMCIA devices will be restricted to using these interrupts (assuming they pass the probe). You may need to use trial and error to find out which interrupts can be safely probed.
- The interrupt probe can be disabled entirely by loading the socket driver with the ``do_scan=0" option. In this case, a default interrupt list will be used, which avoids interrupts already allocated for other devices.

In either case, the probe options can be specified using the `PCIC_OPTS` definition in the PCMCIA startup script, for example:

```
PCIC_OPTS="irq_list=5,9,10"
```

It should be noted that `/proc/interrupts` is completely useless when it comes to diagnosing interrupt probe problems. The probe is sensible enough to never attempt to use an interrupt that is already in use by another Linux driver. So, the PCMCIA drivers are already using all the information in `/proc/interrupts`. Depending on system design, an inactive device can still occupy an interrupt and cause trouble if it is probed for PCMCIA.

3.4 IO port scan failures

Symptoms:

- The system locks up when `cardmgr` is first started, even with no cards present.
- The system log shows a successful host controller probe, including interrupt probe results, but does not show IO probe results.
- In some cases, the IO probe will succeed, but report large numbers of random exclusions.

When `cardmgr` processes IO port ranges listed in `/etc/pcmcia/config.opts`, the kernel probes these ranges to detect latent devices that occupy IO space but are not associated with a Linux driver. The probe is read−only, but in rare cases, reading from a device may interfere with an important system function, resulting in a lock−up.

Your system user's guide may include a map of system devices, showing their IO and memory ranges. These can be explicitly excluded in `config.opts`.

Alternatively, if the probe is unreliable on your system, it can be disabled by setting `CORE_OPTS` to ``probe_io=0''. In this case, you should be very careful to specify only genuinely available ranges of ports in `config.opts`, instead of using the default settings.

3.5 Memory probe failures

Symptoms:

- The core drivers load correctly when no cards are present, with no errors in the system log.
- The system freezes and/or reboots as soon as any card is inserted, before any beeps are heard.

Or alternately:

- All card insertions generate a high beep followed by a low beep.
- All cards are identified as ``anonymous memory cards''.
- The system log reports that various memory ranges have been excluded.

The core modules perform a memory scan at the time of first 16−bit card insertion. This scan can potentially interfere with other memory mapped devices. Also, pre−3.0.0 driver packages perform a more aggressive scan than more recent drivers. The memory window is defined in `/etc/pcmcia/config.opts`. The default window is large, so it may help to restrict the scan to a narrower range. Reasonable ranges to try include 0xd0000−0xdffff, 0xc0000−0xcffff, 0xc8000−0xcffff, or 0xd8000−0xdffff.

If you have DOS or Windows PCMCIA drivers, you may be able to deduce what memory region those drivers use. Note that DOS memory addresses are often specified in ``segment'' form, which leaves off the final hex digit (so an absolute address of 0xd0000 might be given as 0xd000). Be sure to add the extra digit back when making changes to `config.opts`.

In unusual cases, a memory probe failure can indicate a timing register setup problem with the host controller. See the Startup options section for information about dealing with common timing problems.

- `cs: warning: no high memory space available!`

CardBus bridges can allocate memory windows outside of the 640KB−1MB ``memory hole'' in the ISA bus architecture. It is generally a good idea to configure CardBus bridges to use high memory windows, because these are unlikely to conflict with other devices. Also, CardBus cards may require large memory windows, which may be difficult or impossible to fit into low memory. Card Services will preferentially allocate windows in high memory for CardBus bridges, if both low and high memory windows are defined in `config.opts`. The default `config.opts` now includes a high memory window of 0xa0000000−0xa0ffffff. If you have a CardBus bridge and have upgraded from an older PCMCIA driver release, add this memory window if it is not already defined.

In some cases, the default high memory window is not usable. On some IBM Thinkpad models, a window of 0x60000000−0x60ffffff will work in place of the default window.

3.6 Failure to detect card insertions and removals

Symptoms:

- Cards are detected and configured properly if present at boot time.
- The drivers do not respond to insertion and removal events, either by recording events in the system log, or by beeping.

In most cases, the socket driver (`i82365` or `tcic`) will automatically probe and select an appropriate interrupt to signal card status changes. The automatic interrupt probe doesn't work on some Intel−compatible controllers, including Cirrus chips and the chips used in some IBM ThinkPads. If a device is inactive at probe time, its interrupt may also appear to be available. In these cases, the socket driver may pick an interrupt that is used by another device.

With the `i82365` and `tcic` drivers, the `irq_list` option can be used to limit the interrupts that will be tested. This list limits the set of interrupts that can be used by PCMCIA cards as well as for monitoring card status changes. The `cs_irq` option can also be used to explicitly set the interrupt to be used for monitoring card status changes.

If you can't find an interrupt number that works, there is also a polled status mode: both `i82365` and `tcic` will accept a `poll_interval=100` option, to poll for card status changes once per second. This option should also be used if your system has a shortage of interrupts available for use by PCMCIA cards. Especially for systems with more than one host controller, there is little point in dedicating interrupts for monitoring card status changes.

All these options should be set in the `PCIC_OPTS=` line in either `/etc/rc.d/rc.pcmcia` or `/etc/sysconfig/pcmcia`, depending on your site setup.

3.7 Interrupt delivery problems

Symptoms:

- Cards appear to be configured successfully, but don't work.
- Serial and modem cards may respond very sluggishly.
- Network cards may report ``interrupt(s) dropped'', and/or transmit timeouts.

The most simple interrupt delivery problems are due to conflicts with other system devices. These can generally be resolved by excluding problem interrupts in `/etc/pcmcia/config.opts`. To test, just exclude interrupts one by one until either the problem is fixed or you run out of interrupts. If no interrupts work, then device conflicts are probably not the problem.

For CardBus bridges, a variety of other interrupt delivery issues may come into play. For a complete discussion, see PCI interrupt delivery problems.

3.8 System resource starvation

Symptoms:

- When a card is inserted, it is identified correctly but cannot be configured (high/low beep pattern).
- One of the following messages will appear in the system log:

```
RequestIO: Resource in use
RequestIRQ: Resource in use
RequestWindow: Resource in use
GetNextTuple: No more items
could not allocate nn IO ports for CardBus socket n
could not allocate nnK memory for CardBus socket n
could not allocate interrupt for CardBus socket n
```

Interrupt starvation often indicates a problem with the interrupt probe (see Interrupt scan failures). In some cases, the probe will seem to work, but only report one or two available interrupts. Check your system log to see if the scan results look sensible. Disabling the probe and selecting interrupts manually should help.

If the interrupt probe is not working properly, the socket driver may allocate an interrupt for monitoring card insertions, even when interrupts are too scarce for this to be a good idea. You can switch the controller to polled mode by setting `PCIC_OPTS` to ``poll_interval=100'. Or, if you have a CardBus controller and an older version of the PCMCIA drivers, try ``pci_csc=1", which selects a PCI interrupt (if available) for card status changes.

IO port starvation is fairly uncommon, but sometimes happens with cards that require large, contiguous, aligned regions of IO port space, or that only recognize a few specific IO port positions. The default IO port ranges in `/etc/pcmcia/config.opts` are normally sufficient, but may be extended. If this is the problem, try uncommenting the ``include port 0x1000-0x17ff" line in `config.opts`. In rare cases, starvation may indicate that the IO port probe failed (see IO port scan failures).

Memory starvation is also uncommon with the default memory window settings in `config.opts`. CardBus cards may require larger memory regions than typical 16−bit cards. Since CardBus memory windows can be mapped anywhere in the host's PCI address space (rather than just in the 640K−1MB ``hole" in PC systems), it is helpful to specify large memory windows in high memory, such as 0xa0000000−0xa0ffffff.

3.9 Resource conflict only with two cards inserted

Symptoms:

- Two cards each work fine when used separately.
- When both cards are inserted, only one works.

This usually indicates a resource conflict with a system device that Linux does not know about. PCMCIA devices are dynamically configured, so, for example, interrupts are allocated as needed, rather than specifically assigned to particular cards or sockets. Given a list of resources that appear to be available, cards are assigned resources in the order they are configured. In this case, the card configured last is being assigned a resource that in fact is not free.

Check the system log to see what resources are used by the non−working card. Exclude these in `/etc/pcmcia/config.opts`, and restart the `cardmgr` daemon to reload the resource database.

3.10 Device configuration does not complete

Symptoms:

- When a card is inserted, exactly one high beep is heard.
- Subsequent card insertions and removals may be ignored.

This indicates that the card was identified successfully, however, `cardmgr` has been unable to complete the configuration process for some reason. The most likely reason is that a step in the card setup script has blocked. A good example would be the network script blocking if a network card is inserted with no actual network hookup present.

To pinpoint the problem, you can manually run a setup script to see where it is blocking. The scripts are in the `/etc/pcmcia` directory. They take two parameters: a device name, and an action. The `cardmgr` daemon records the configuration commands in the system log. For example, if the system log shows that the command ``./network start eth0'' was the last command executed by `cardmgr`, the following command would trace the script:

```
sh −x /etc/pcmcia/network start eth0
```

4.Usage and features

4.1 Tools for configuring and monitoring PCMCIA devices

If the modules are all loaded correctly, the output of the `lsmod` command should look like the following, when no cards are inserted:

```
Module                  Size  Used by
ds                      5640   2
```

```
i82365                 15452   2
pcmcia_core            30012   3  [ds i82365]
```

The system log should also include output from the socket driver describing the host controller(s) found and the number of sockets detected.

The cardmgr configuration daemon

The `cardmgr` daemon is responsible for monitoring PCMCIA sockets, loading client drivers when needed, and running user–level scripts in response to card insertions and removals. It records its actions in the system log, but also uses beeps to signal card status changes. The tones of the beeps indicate success or failure of particular configuration steps. Two high beeps indicate that a card was identified and configured successfully. A high beep followed by a low beep indicates that a card was identified, but could not be configured for some reason. One low beep indicates that a card could not be identified.

The `cardmgr` daemon configures cards based on a database of known card types kept in `/etc/pcmcia/config`. This file describes the various client drivers, then describes how to identify various cards, and which driver(s) belong with which cards. The format of this file is described in the `pcmcia(5)` man page.

The socket status file, stab

`Cardmgr` records device information for each socket in `/var/lib/pcmcia/stab`. Here is a sample `stab` listing:

```
Socket 0: Adaptec APA-1460 SlimSCSI
0       scsi    aha152x_cs      0       sda     8       0
0       scsi    aha152x_cs      1       scd0    11      0
Socket 1: Serial or Modem Card
1       serial  serial_cs       0       ttyS1   5       65
```

For the lines describing devices, the first field is the socket, the second is the device class, the third is the driver name, the fourth is used to number multiple devices associated with the same driver, the fifth is the device name, and the final two fields are the major and minor device numbers for this device (if applicable). See the `stab` man page for more info.

The cardctl and cardinfo utilities

The `cardctl` command can be used to check the status of a socket, or to see how it is configured. It can also be used to alter the configuration status of a card. Here is an example of the output of the ``` ``cardctl config'' ``` command:

```
Socket 0:
  not configured
Socket 1:
  Vcc = 5.0, Vpp1 = 0.0, Vpp2 = 0.0
  Card type is memory and I/O
  IRQ 3 is dynamic shared, level mode, enabled
  Speaker output is enabled
  Function 0:
    Config register base = 0x0800
      Option = 0x63, status = 0x08
    I/O window 1: 0x0280 to 0x02bf, auto sized
    I/O window 2: 0x02f8 to 0x02ff, 8 bit
```

Or ``cardctl ident'', to get card identification information:

```
Socket 0:
  no product info available
Socket 1:
  product info: "LINKSYS", "PCMLM336", "A", "0040052D6400"
  manfid: 0x0143, 0xc0ab
  function: 0 (multifunction)
```

The ``cardctl suspend'' and ``cardctl resume'' commands can be used to shut down a card without unloading its associated drivers. The ``cardctl reset'' command attempts to reset and reconfigure a card. ``cardctl insert'' and ``cardctl eject'' mimic the actions performed when a card is physically inserted or ejected, including loading or unloading drivers, and configuring or shutting down devices.

If you are running X, the cardinfo utility produces a graphical display showing the current status of all PCMCIA sockets, similar in content to ``cardctl config''. It also provides a graphical interface to most other cardctl functions.

Inserting and ejecting cards

In theory, you can insert and remove PCMCIA cards at any time. However, it is a good idea not to eject a card that is currently being used by an application program. Kernels older than 1.1.77 would often lock up when serial/modem cards were ejected, but this should be fixed now.

Some card types cannot be safely hot ejected. Specifically, ATA/IDE and SCSI interface cards are not hot−swap−safe. This is unlikely to be fixed, because a complete solution would require significant changes to the Linux block device model. Also, it is generally not safe to hot eject CardBus cards of any type. This is likely to improve gradually as hot swap bugs in the CardBus drivers are found and fixed. For these card types (IDE, SCSI, CardBus), it is recommended that you always use ``cardctl eject'' before ejecting.

Card Services and Advanced Power Management

Card Services can be compiled with support for APM (Advanced Power Management) if you've configured your kernel with APM support. The APM kernel driver is maintained by Stephen Rothwell (Stephen.Rothwell@canb.auug.org.au). The `apmd` daemon is maintained by Avery Pennarun (apenwarr@worldvisions.ca), with more information available at http://www.worldvisions.ca/~apenwarr/apmd/. The PCMCIA modules will automatically be configured for APM if a compatible version is detected on your system.

Whether or not APM is configured, you can use ``` ``cardctl suspend'' ``` before suspending your laptop, and ``` ``cardctl resume'' ``` after resuming, to cleanly shut down and restart your PCMCIA cards. This will not work with a modem that is in use, because the serial driver isn't able to save and restore the modem operating parameters.

APM seems to be unstable on some systems. If you experience trouble with APM and PCMCIA on your system, try to narrow down the problem to one package or the other before reporting a bug.

Some drivers, notably the PCMCIA SCSI drivers, cannot recover from a suspend/resume cycle. When using a PCMCIA SCSI card, always use ``` ``cardctl eject'' ``` prior to suspending the system.

Shutting down the PCMCIA system

To unload the entire PCMCIA package, invoke `rc.pcmcia` with:

```
        /etc/rc.d/rc.pcmcia stop
```

This script will take several seconds to run, to give all client drivers time to shut down gracefully. If a device is currently in use, the shutdown will be incomplete, and some kernel modules may not be unloaded. To avoid this, use ``` ``cardctl eject'' ``` to shut down all sockets before invoking `rc.pcmcia`. The exit status of the `cardctl` command will indicate if any sockets could not be shut down.

4.2 Overview of the PCMCIA configuration scripts

Each PCMCIA device has an associated ``class'' that describes how it should be configured and managed. Classes are associated with device drivers in `/etc/pcmcia/config`. There are currently five IO device classes (network, SCSI, cdrom, fixed disk, and serial) and two memory device classes (memory and FTL). For each class, there are two scripts in `/etc/pcmcia`: a main configuration script (i.e., `/etc/pcmcia/scsi` for SCSI devices), and an options script (i.e., `/etc/pcmcia/scsi.opts`). The main script for a device will be invoked to configure that device when a card is inserted, and to shut down the device when the card is removed. For cards with several associated devices, the script will be invoked for each device.

The config scripts start by extracting some information about a device from the `stab` file. Each script constructs a ``device address'', that uniquely describes the device it has been asked to configure, in the

ADDRESS shell variable. This is passed to the *.opts script, which should return information about how a device at this address should be configured. For some devices, the device address is just the socket number. For others, it includes extra information that may be useful in deciding how to configure the device. For example, network devices pass their hardware ethernet address as part of the device address, so the network.opts script could use this to select from several different configurations.

The first part of all device addresses is the current PCMCIA ``scheme''. This parameter is used to support multiple sets of device configurations based on a single external user−specified variable. One use of schemes would be to have a ``home'' scheme, and a ``work'' scheme, which would include different sets of network configuration parameters. The current scheme is selected using the ``cardctl scheme'' command. The default if no scheme is set is ``default''.

As a general rule, when configuring Linux for a laptop, PCMCIA devices should only be configured from the PCMCIA device scripts. Do not try to configure a PCMCIA device the same way you would configure a permanently attached device. However, some Linux distributions provide PCMCIA packages that are hooked into those distributions' own device configuration tools. In that case, some of the following sections may not apply; ideally, this will be documented by the distribution maintainers.

4.3 PCMCIA network adapters

Linux ethernet−type network interfaces normally have names like eth0, eth1, and so on. Token−ring adapters are handled similarly, however they are named tr0, tr1, and so on. The ifconfig command is used to view or modify the state of a network interface. A peculiarity of Linux is that network interfaces do not have corresponding device files under /dev, so do not be surprised when you do not find them.

When an ethernet card is detected, it will be assigned the first free interface name, which will normally be eth0. Cardmgr will run the /etc/pcmcia/network script to configure the interface, which normally reads network settings from /etc/pcmcia/network.opts. The network and network.opts scripts will be executed only when your ethernet card is actually present. If your system has an automatic network configuration facility, it may or may not be PCMCIA−aware. Consult the documentation of your Linux distribution and the Notes about specific Linux distributions to determine if PCMCIA network devices should be configured with the automatic tools, or by editing network.opts.

The device address passed to network.opts consists of four comma−separated fields: the scheme, the socket number, the device instance, and the card's hardware ethernet address. The device instance is used to number devices for cards that have several network interfaces, so it will usually be 0. If you have several network cards used for different purposes, one option would be to configure the cards based on socket position, as in:

```
case "$ADDRESS" in
*,0,*,*)
    # definitions for network card in socket 0
    ;;
*,1,*,*)
    # definitions for network card in socket 1
    ;;
esac
```

Alternatively, they could be configured using their hardware addresses, as in:

```
case "$ADDRESS" in
*,*,*,00:80:C8:76:00:B1)
    # definitions for a D-Link card
    ;;
*,*,*,08:00:5A:44:80:01)
    # definitions for an IBM card
esac
```

Network device parameters

The following parameters can be defined in `network.opts`:

IF_PORT

Specifies the ethernet transceiver type, for certain 16–bit cards that do not autodetect. See ``man ifport'' for more information.

BOOTP

A boolean (y/n) value: indicates if the host's IP address and routing information should be obtained using the BOOTP protocol, with `bootpc` or `pump`.

DHCP

A boolean (y/n) value: indicates if the host's IP address and routing information should be obtained from a DHCP server. The network script first looks for `dhcpcd`, then `dhclient`, then `pump`.

DHCP_HOSTNAME

Specifies a hostname to be passed to `dhcpcd` or `pump`, for inclusion in DHCP messages.

IPADDR

The IP address for this interface.

NETMASK, BROADCAST, NETWORK

Basic network parameters: see the networking HOWTO for more information.

GATEWAY

The IP address of a gateway for this host's subnet. Packets with destinations outside this subnet will be routed to this gateway.

DOMAIN

The local network domain name for this host, to be used in creating `/etc/resolv.conf`.

SEARCH

A search list for host name lookup, to be added to `/etc/resolv.conf`. `DOMAIN` and `SEARCH` are mutually exclusive: see ``` ``man resolver'' ``` for more information.

DNS_1, DNS_2, DNS_3

Host names or IP addresses for nameservers for this interface, to be added to `/etc/resolv.conf`

MOUNTS

A space−separated list of NFS mount points to be mounted for this interface.

IPX_FRAME, IPX_NETNUM

For IPX networks: the frame type and network number, passed to the `ipx_interface` command.

NO_CHECK, NO_FUSER

Boolean (y/n) settings for card eject policy. If `NO_CHECK` is set, then ``` ``cardctl eject'' ``` will shut down a device even if there are open connections. If `NO_FUSER` is set, then the script will not check for busy NFS mounts or kill processes using those mounts.

For example:

```
case "$ADDRESS" in
*,*,*,*)
    IF_PORT="10base2"
    BOOTP="n"
    IPADDR="10.0.0.1"
    NETMASK="255.255.255.0"
    NETWORK="10.0.0.0"
    BROADCAST="10.0.0.255"
    GATEWAY="10.0.0.1"
    DOMAIN="domain.org"
    DNS_1="dns1.domain.org"
    ;;
esac
```

To automatically mount and unmount NFS filesystems, first add all these filesystems to `/etc/fstab`, but include `noauto` in the mount options. In `network.opts`, list the filesystem mount points in the `MOUNTS` variable. It is especially important to use either `cardctl` or `cardinfo` to shut down a network card when NFS mounts are active. It is not possible to cleanly unmount NFS filesystems if a network card is simply ejected without warning.

In addition to the usual network configuration parameters, the network.opts script can specify extra actions to be taken after an interface is configured, or before an interface is shut down. If network.opts defines a shell function called start_fn, it will be invoked by the network script after the interface is configured, and the interface name will be passed to the function as its first (and only) argument. Similarly, if it is defined, stop_fn will be invoked before shutting down an interface.

The transceiver type for some cards can be selected using the IF_PORT setting. This can either be a numeric value, or a keyword identifying the transceiver type. All the network drivers default to either autodetect the interface if possible, or 10baseT otherwise. The ifport command can be used to check or set the current transceiver type. For example:

```
# ifport eth0 10base2
#
# ifport eth0
eth0    2 (10base2)
```

The current (3.0.10 or later) 3c589 driver should quickly autodetect transceiver changes at any time. Earlier releases of the 3c589 driver had a somewhat slow and flaky transceiver autodetection algorithm. For these releases, the appropriate network cable should be connected to the card when the card is configured, or you can force autodetection with:

```
ifconfig eth0 down up
```

Comments about specific cards

- With IBM CCAE and Socket EA cards, the transceiver type (10base2, 10baseT, AUI) needs to be set when the network device is configured. Make sure that the transceiver type reported in the system log matches your connection.
- The Farallon EtherWave is actually based on the 3Com 3c589, with a special transceiver. Though the EtherWave uses 10baseT−style connections, its transceiver requires that the 3c589 be configured in 10base2 mode.
- If you have trouble with an IBM CCAE, NE4100, Thomas Conrad, or Kingston adapter, try increasing the memory access time with the mem_speed=# option to the pcnet_cs module. An example of how to do this is given in the standard config.opts file. Try speeds of up to 1000 (in nanoseconds).
- For the New Media Ethernet adapter, on some systems, it may be necessary to increase the IO port access time with the io_speed=# option when the pcmcia_core module is loaded. Edit CORE_OPTS in the startup script to set this option.
- The multicast support in the New Media Ethernet driver is incomplete. The latest driver will function with multicast kernels, but will ignore multicast packets. Promiscuous mode should work properly.
- The driver used by the IBM and 3Com token ring adapters seems to behave very badly if the cards are not connected to a ring when they get initialized. Always connect these cards to the net before

they are powered up. If `ifconfig` reports the hardware address as all 0's, this is likely to be due to a memory window configuration problem.
- Some Linksys, D−Link, and IC−Card 10baseT/10base2 cards have a unique way of selecting the transceiver type that isn't handled by the Linux drivers. One workaround is to boot DOS and use the vendor−supplied utility to select the transceiver, then warm boot Linux. Alternatively, a Linux utility to perform this function is available at ftp://projects.sourceforge.net/pub/pcmcia−cs/extras/dlport.c.
- 16−bit PCMCIA cards have a maximum performance of 1.5−2 MB/sec. That means that any 16−bit 100baseT card (i.e., any card that uses the `pcnet_cs`, `3c574_cs`, `smc91c92_cs`, or `xirc2ps_cs` driver) will never achieve full 100baseT throughput. Only CardBus network adapters can fully exploit 100baseT data rates.
- For WaveLAN wireless network adapters, Jean Tourrilhes (`jt@hpl.hp.com`) has put together a wireless HOWTO at http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/.

Diagnosing problems with network adapters

- In 3.1.15 and later PCMCIA releases, the `test_network` script in the `debug−tools` subdirectory of the PCMCIA source tree will spot some common problems.
- Is your card recognized as an ethernet card? Check the system log and make sure that `cardmgr` identifies the card correctly and starts up one of the network drivers. If it doesn't, your card might still be usable if it is compatible with a supported card. This will be most easily done if the card claims to be ``NE2000 compatible''.
- Is the card configured properly? If you are using a supported card, and it was recognized by `cardmgr`, but still doesn't work, there might be an interrupt or port conflict with another device. Find out what resources the card is using (from the system log), and try excluding these in `/etc/pcmcia/config.opts` to force the card to use something different.
- If your card seems to be configured properly, but sometimes locks up, particularly under high load, you may need to try changing your socket driver timing parameters. See the Startup options section for more information.
- If you get ``Network is unreachable'' messages when you try to access the network, then the routing information specified in `/etc/pcmcia/network.opts` is incorrect. This exact message is an absolutely foolproof indication of a routing error. On the other hand, mis−configured cards will usually fail silently.
- If you are trying to use DHCP to configure your network interface, try testing things with a static IP address instead, to rule out a DHCP configuration problem.
- To diagnose problems in `/etc/pcmcia/network.opts`, start by trying to ping other systems on the same subnet using their IP addresses. Then try to ping your gateway, and then machines on other subnets. Ping machines by name only after trying these simpler tests.
- Make sure your problem is really a PCMCIA one. It may help to see see if the card works under DOS with the vendor's drivers. Double check your modifications to the `/etc/pcmcia/network.opts` script. Make sure your drop cable, ``T'' jack, terminator, etc are working.
- Use real network cables. Don't even think about using that old phone cord you found in your basement. And this means Category 5 cable for 100baseT. It really matters.

4.4 PCMCIA serial and modem devices

Linux serial devices are accessed via the `/dev/ttyS*` and `/dev/cua*` special device files. In pre–2.2 kernels, the `ttyS*` devices were for incoming connections, such as directly connected terminals, and the `cua*` devices were for outgoing connections, such as modems. Use of `cua*` devices is deprecated in current kernels, and `ttyS*` can be used for all applications. The configuration of a serial device can be examined and modified with the `setserial` command.

When a serial or modem card is detected, it will be assigned to the first available serial device slot. This will usually be `/dev/ttyS1` (`cua1`) or `/dev/ttyS2` (`cua2`), depending on the number of built–in serial ports. The `ttyS*` device is the one reported in `stab`. The default serial device option script, `/etc/pcmcia/serial.opts`, will link the device file to `/dev/modem` as a convenience. For pre–2.2 kernels, the link is made to the `cua*` device.

Do not try to use `/etc/rc.d/rc.serial` to configure a PCMCIA modem. This script should only be used to configure non–removable devices. Modify `/etc/pcmcia/serial.opts` if you want to do anything special to set up your modem. Also, do not try to change the IO port and interrupt settings of a serial device using `setserial`. This would tell the serial driver to look for the device in a different place, but would not change how the card's hardware is actually configured. The serial configuration script allows you to specify other `setserial` options, as well as whether a line should be added to `/etc/inittab` for this port.

The device address passed to `serial.opts` has three comma–separated fields: the first is the scheme, the second is the socket number, and the third is the device instance. The device instance may take several values for cards that support multiple serial ports, but for single–port cards, it will always be 0. If you commonly use more than one modem, you may want to specify different settings based on socket position, as in:

```
case "$ADDRESS" in
*,0,*)
    # Options for modem in socket 0
    LINK=/dev/modem0
    ;;
*,1,*)
    # Options for modem in socket 1
    LINK=/dev/modem1
    ;;
esac
```

If a PCMCIA modem is already configured when Linux boots, it may be incorrectly identified as an ordinary built–in serial port. This is harmless, however, when the PCMCIA drivers take control of the modem, it will be assigned a different device slot. It is best to either parse `stab` or use `/dev/modem`, rather than expecting a PCMCIA modem to always have the same device assignment.

If you configure your kernel to load the basic Linux serial port driver as a module, you must edit `/etc/pcmcia/config` to indicate that this module must be loaded. Edit the serial device entry to read:

```
device "serial_cs"
  class "serial" module "misc/serial", "serial_cs"
```

Serial device parameters

The following parameters can be defined in `serial.opts`:

LINK

Specifies a path for a symbolic link to be created to the ``callout" device (e.g., `/dev/cua*` for pre–2.2, or `/dev/ttyS*` for 2.2 kernels).

SERIAL_OPTS

Specifies options to be passed to the `setserial` command.

INITTAB

If specified, this will be used to construct an `inittab` entry for the device.

NO_CHECK, NO_FUSER

Boolean (y/n) settings for card eject policy. If `NO_CHECK` is true, then ``cardctl eject" will shut down a device even if it is busy. If `NO_FUSER` is true, then the script will not try to kill processes using an ejected device.

For example:

```
case "$ADDRESS" in
*,*,*,*)
    LINK="/dev/modem"
    SERIAL_OPTS=""
    INITTAB="/sbin/getty"
```

Comments about specific cards

- The Uniden Data 2000 Wireless CDPD card has some special dialing strings for initiating SLIP and PPP mode. For SLIP, use ``ATDT2"; for PPP, "ATDT0".

Diagnosing problems with serial devices

- In 3.1.15 and later PCMCIA releases, the `test_modem` script in the `debug-tools` subdirectory of the PCMCIA source tree will spot some common problems.
- Is your card recognized as a modem? Check the system log and make sure that `cardmgr` identifies

the card correctly and starts up the `serial_cs` driver. If it doesn't, you may need to add a new entry to your `/etc/pcmcia/config` file so that it will be identified properly. See the Configuring unrecognized cards section for details.
- Is the modem configured successfully by serial_cs? Again, check the system log and look for messages from the serial_cs driver. If you see ``register_serial() failed'', you may have an I/O port conflict with another device. Another tip−off of a conflict is if the device is reported to be an 8250; most modern modems should be identified as 16550A UART's. If you think you're seeing a port conflict, edit `/etc/pcmcia/config.opts` and exclude the port range that was allocated for the modem.
- Is there an interrupt conflict? If the system log looks good, but the modem just doesn't seem to work, try using `setserial` to change the irq to 0, and see if the modem works. This causes the serial driver to use a slower polled mode instead of using interrupts. If this seems to fix the problem, it is likely that some other device in your system is using the interrupt selected by serial_cs. You should add a line to `/etc/pcmcia/config.opts` to exclude this interrupt.
- If the modem seems to work only very, very slowly, this is an almost certain indicator of an interrupt conflict.
- Make sure your problem is really a PCMCIA one. It may help to see if the card works under DOS with the vendor's drivers. Also, don't test the card with something complex like SLIP or PPP until you are sure you can make simple connections. If simple things work but SLIP does not, your problem is most likely with SLIP, not with PCMCIA.
- If you get kernel messages indicating that the serial_cs module cannot be loaded, it means that your kernel does not have serial device support. If you have compiled the serial driver as a module, you must modify `/etc/pcmcia/config` to indicate that the `serial` module should be loaded before `serial_cs`.

4.5 PCMCIA parallel port devices

The Linux parallel port driver is layered so that several high−level device types can share use of the same low level port driver. Printer devices are accessed via the `/dev/lp*` special device files. The configuration of a printer device can be examined and modified with the `tunelp` command.

The `parport_cs` module depends on the `parport` and `parport_pc` drivers, which may be either compiled into the kernel or compiled as modules. The layered driver structure means that any top−level parallel drivers (such as the plip driver, the printer driver, etc) must be compiled as modules. These drivers only recognize parallel port devices at module startup time, so they need to be loaded after any PC Card parallel devices are configured.

The device address passed to `parport.opts` has three comma−separated fields: the first is the scheme, the second is the socket number, and the third is the device instance. The device instance may take several values for cards that support multiple parallel ports, but for single−port cards, it will always be 0. If you commonly use more than one such card, you may want to specify different settings based on socket position, as in:

```
case "$ADDRESS" in
*,0,*)
    # Options for card in socket 0
    LINK=/dev/printer0
    ;;
*,1,*)
```

```
    # Options for card in socket 1
    LINK=/dev/printer1
    ;;
esac
```

If you configure your kernel to load the basic Linux parallel port driver as a module, you must edit `/etc/pcmcia/config` to indicate that the appropriate modules must be loaded. Edit the parallel device entry to read:

```
device "parport_cs"
  class "parport" module "misc/parport", "misc/parport_pc", "parport_cs"
```

Parallel device parameters

The following parameters can be defined in `parport.opts`:

LINK

Specifies a path for a symbolic link to be created to the printer port.

LP_OPTS

Specifies options to be passed to the `tunelp` command.

NO_CHECK, NO_FUSER

Boolean (y/n) settings for card eject policy. If `NO_CHECK` is true, then ``` ``cardctl eject'' ``` will shut down a device even if it is busy. If `NO_FUSER` is true, then the script will not try to kill processes using an ejected device.

For example:

```
case "$ADDRESS" in
*,*,*,*)
    LINK="/dev/printer"
    LP_OPTS=""
```

Diagnosing problems with parallel port devices

- Is there an interrupt conflict? If the system log looks good, but the port just doesn't seem to work, try using `tunelp` to change the irq to 0, and see if things improve. This switches the driver to polling

mode. If this seems to fix the problem, it is likely that some other device in your system is using the interrupt selected by parport_cs. You should add a line to `/etc/pcmcia/config.opts` to exclude this interrupt.
- If you get kernel messages indicating that the `parport_cs` module cannot be loaded, it means that your kernel does not have parallel device support. If you have compiled the parallel driver as a module, you may need to modify `/etc/pcmcia/config` to indicate that the `parport` and `parport_pc` modules should be loaded before `parport_cs`.

4.6 PCMCIA SCSI adapters

All the currently supported PCMCIA SCSI cards are work−alikes of one of the following ISA bus cards: the Qlogic, the Adaptec AHA−152X, or the Future Domain TMC−16x0. The PCMCIA drivers are built by linking some PCMCIA−specific code (in `qlogic_cs.c`, `aha152x_cs.c`, or `fdomain_cs.c`) with the normal Linux SCSI driver, pulled from the Linux kernel source tree. The Adaptec APA1480 CardBus driver is based on the kernel aic7xxx PCI driver. Due to limitations in the Linux SCSI driver model, only one removable card per driver is supported.

When a new SCSI host adapter is detected, the SCSI drivers will probe for devices. Check the system log to make sure your devices are detected properly. New SCSI devices will be assigned to the first available SCSI device files. The first SCSI disk will be `/dev/sda`, the first SCSI tape will be `/dev/st0`, and the first CD−ROM will be `/dev/scd0`.

A list of SCSI devices connected to this host adapter will be shown in `stab`, and the SCSI configuration script, `/etc/pcmcia/scsi`, will be called once for each attached device, to either configure or shut down that device. The default script does not take any actions to configure SCSI devices, but will properly unmount filesystems on SCSI devices when a card is removed.

The device addresses passed to `scsi.opts` are complicated, because of the variety of things that can be attached to a SCSI adapter. Addresses consist of either six or seven comma−separated fields: the current scheme, the device type, the socket number, the SCSI channel, ID, and logical unit number, and optionally, the partition number. The device type will be ``sd'' for disks, ``st'' for tapes, ``sr'' for CD−ROM devices, and ``sg'' for generic SCSI devices. For most setups, the SCSI channel and logical unit number will be 0. For disk devices with several partitions, `scsi.opts` will first be called for the whole device, with a five−field address. The script should set the `PARTS` variable to a list of partitions. Then, `scsi.opts` will be called for each partition, with the longer seven−field addresses.

If your kernel does not have a top−level driver (disk, tape, etc) for a particular SCSI device, then the device will not be configured by the PCMCIA drivers. As a side effect, the device's name in `stab` will be something like ``sd#nnnn'' where ``nnnn'' is a four−digit hex number. This happens when `cardmgr` is unable to translate a SCSI device ID into a corresponding Linux device name.

It is possible to modularize the top−level SCSI drivers so that they are loaded on demand. To do so, you need to edit `/etc/pcmcia/config` to tell `cardmgr` which extra modules need to be loaded when your adapter is configured. For example:

```
device "aha152x_cs"
  class "scsi" module "scsi/scsi_mod", "scsi/sd_mod", "aha152x_cs"
```

would say to load the core SCSI module and the top–level disk driver module before loading the regular PCMCIA driver module. The PCMCIA Configure script will not automatically detect modularized SCSI modules, so you will need use the manual configure option to enable SCSI support.

Always turn on SCSI devices before powering up your laptop, or before inserting the adapter card, so that the SCSI bus is properly terminated when the adapter is configured. Also be very careful about ejecting a SCSI adapter. Be sure that all associated SCSI devices are unmounted and closed before ejecting the card. The best way to ensure this is to use either `cardctl` or `cardinfo` to request card removal before physically ejecting the card. For now, all SCSI devices should be powered up before plugging in a SCSI adapter, and should stay connected until after you unplug the adapter and/or power down your laptop.

There is a potential complication when using these cards that does not arise with ordinary ISA bus adapters. The SCSI bus carries a ``termination power'' signal that is necessary for proper operation of ordinary passive SCSI terminators. PCMCIA SCSI adapters do not supply termination power, so if it is required, an external device must supply it. Some external SCSI devices may be configured to supply termination power. Others, such as the Zip Drive and the Syquest EZ–Drive, use active terminators that do not depend on it. In some cases, it may be necessary to use a special terminator block such as the APS SCSI Sentry 2, which has an external power supply. When configuring your SCSI device chain, be aware of whether or not any of your devices require or can provide termination power.

SCSI device parameters

The following parameters can be defined in `scsi.opts`:

LINK

Specifies a path for a symbolic link to be created to this device.

DO_FSTAB

A boolean (y/n) setting: specifies if an entry should be added to `/etc/fstab` for this device.

DO_FSCK

A boolean (y/n) setting: specifies if the filesystem should be checked before being mounted, with ``fsck –Ta''.

DO_MOUNT

A boolean (y/n) setting: specifies if this device should be automatically mounted at card insertion time.

FSTYPE, OPTS, MOUNTPT

The filesystem type, mount options, and mount point to be used for the fstab entry and/or mounting the device.

NO_CHECK, NO_FUSER

Boolean (y/n) settings for card eject policy. If `NO_CHECK` is true, then ``cardctl eject'' will shut down a device even if it is busy. If `NO_FUSER` is true, then the script will not try to kill processes using an ejected device.

For example, here is a script for configuring a disk device at SCSI ID 3, with two partitions, and a CD−ROM at SCSI ID 6:

```
case "$ADDRESS" in
*,sd,*,0,3,0)
    # This device has two partitions...
    PARTS="1 2"
    ;;
*,sd,*,0,3,0,1)
    # Options for partition 1:
    #  update /etc/fstab, and mount an ext2 fs on /usr1
    DO_FSTAB="y" ; DO_FSCK="y" ; DO_MOUNT="y"
    FSTYPE="ext2"
    OPTS=""
    MOUNTPT="/usr1"
    ;;
*,sd,*,0,3,0,2)
    # Options for partition 2:
    #  update /etc/fstab, and mount an MS-DOS fs on /usr2
    DO_FSTAB="y" ; DO_FSCK="y" ; DO_MOUNT="y"
    FSTYPE="msdos"
    OPTS=""
    MOUNTPT="/usr2"
    ;;
*,sr,*,0,6,0)
    # Options for CD-ROM at SCSI ID 6
    PARTS=""
    DO_FSTAB="y" ; DO_FSCK="n" ; DO_MOUNT="y"
    FSTYPE="iso9660"
    OPTS="ro"
    MOUNTPT="/cdrom"
    ;;
esac
```

Comments about specific cards

- The Adaptec APA−1480 CardBus card needs a large IO port window (256 contiguous ports aligned on a 256−port boundary). It may be necessary to expand the IO port regions in `/etc/pcmcia/config.opts` to guarantee that such a large window can be found.
- The Adaptec APA−460 SlimSCSI adapter is not supported. This card was originally sold under the Trantor name, and when Adaptec merged with Trantor, they continued to sell the Trantor card with an Adaptec label. The APA−460 is not compatible with any existing Linux driver.
- I have had one report of a bad interaction between the New Media Bus Toaster and a UMAX Astra 1200s scanner. Due to the complexity of the SCSI protocol, when diagnosing problems with SCSI devices, it is worth considering that incompatible combinations like this may exist and may not be documented.

Diagnosing problems with SCSI adapters

* With the aha152x_cs driver (used by Adaptec, New Media, and a few others), it seems that SCSI disconnect/reconnect support is a frequent source of trouble with tape drives. To disable this ``feature,'' add the following to /etc/pcmcia/config.opts:

```
module "aha152x_cs" opts "reconnect=0"
```

* Also with the aha152x_cs driver, certain devices seem to require a longer startup delay, controlled via the reset_delay module parameter. The Yamaha 4416S CDR drive is one such device. The result is the device is identified successfully, then hangs the system. In such cases, try:

```
module "aha152x_cs" opts "reset_delay=500"
```

* Another potential source of SCSI device probe problems is probing of multiple LUN's. If you see successful detection of a device, followed by SCSI bus timeouts when LUN 1 for that device is probed, then disable the kernel's CONFIG_SCSI_MULTI_LUN option.
* If you have compiled SCSI support as modules (CONFIG_SCSI is ``m''), you must modify /etc/pcmcia/config to load the SCSI modules before the appropriate *_cs driver is loaded.
* If you get ``aborting command due to timeout'' messages when the SCSI bus is probed, you almost certainly have an interrupt conflict.
* If the host driver reports ``no SCSI devices found'', verify that your kernel was compiled with the appropriate top−level SCSI drivers for your devices (i.e., disk, tape, CD−ROM, and/or generic). If a top−level driver is missing, devices of that type will be ignored.

4.7 PCMCIA memory cards

The memory_cs driver handles all types of memory cards, as well as providing direct access to the PCMCIA memory address space for cards that have other functions. When loaded, it creates a combination of character and block devices. See the man page for the module for a complete description of the device naming scheme. Block devices are used for disk−like access (creating and mounting filesystems, etc). The character devices are for "raw" unbuffered reads and writes at arbitrary locations.

The device address passed to memory.opts consists of two fields: the scheme, and the socket number. The options are applied to the first common memory partition on the corresponding memory card.

Some flash memory cards, and most simple static RAM cards, lack a ``Card Information Structure'' (CIS), which is the system PCMCIA cards use to identify themselves. Normally, cardmgr will assume that any card that lacks a CIS is a simple memory card, and load the memory_cs driver. Thus, a common side effect of a general card identification problem is that other types of cards may be misdetected as memory cards.

There is another issue to consider when handling memory cards that do not have CIS information. At startup time, the PCMCIA package tries to use the first detected card to determine what memory regions are usable for PCMCIA. The memory scan can be fooled if that card is a simple memory card. If you plan to use memory cards often, it is best to limit the memory windows in /etc/pcmcia/config.opts to known−good regions.

The memory_cs driver uses a heuristic to guess the capacity of these cards. The heuristic does not work for write protected cards, and may make mistakes in some other cases as well. If a card is misdetected, its size should then be explicitly specified when using commands such as dd or mkfs. The memory_cs module also has a parameter for overriding the size detection. See the man page.

Memory device parameters

The following parameters can be specified in memory.opts:

DO_FSTAB

A boolean (y/n) setting: specifies if an entry should be added to /etc/fstab for this device.

DO_FSCK

A boolean (y/n) setting: specifies if the filesystem should be checked before being mounted, with ``fsck -Ta''.

DO_MOUNT

A boolean (y/n) setting: specifies if this device should be automatically mounted at card insertion time.

FSTYPE, OPTS, MOUNTPT

The filesystem type, mount options, and mount point to be used for the fstab entry and/or mounting the device.

NO_CHECK, NO_FUSER

Boolean (y/n) settings for card eject policy. If NO_CHECK is true, then ``cardctl eject'' will shut down a device even if it is busy. If NO_FUSER is true, then the script will not try to kill processes using an ejected device.

Here is an example of a script that will automatically mount memory cards based on which socket they are inserted into:

```
case "$ADDRESS" in
*,0,0)
    # Mount filesystem, but don't update /etc/fstab
    DO_FSTAB="n" ; DO_FSCK="y" ; DO_MOUNT="y"
    FSTYPE="ext2" ; OPTS=""
    MOUNTPT="/mem0"
    ;;
*,1,0)
    # Mount filesystem, but don't update /etc/fstab
```

```
    DO_FSTAB="n" ; DO_FSCK="y" ; DO_MOUNT="y"
    FSTYPE="ext2" ; OPTS=""
    MOUNTPT="/mem1"
    ;;
esac
```

Using linear flash memory cards

The following information applies only to so−called ``linear flash'' memory cards. Many flash cards, including all SmartMedia and CompactFlash cards, actually include circuitry to emulate an IDE disk device. Those cards are thus handled as IDE devices, not memory cards.

There are two major formats for flash memory cards: the FTL or ``flash translation layer'' style, and the Microsoft Flash File System. The FTL format is generally more flexible because it allows any ordinary high−level filesystem (ext2, ms−dos, etc) to be used on a flash card as if it were an ordinary disk device. The FFS is a completely different filesystem type. Linux cannot currently handle cards formated with FFS.

To use a flash memory card as an ordinary disk−like block device, first create an FTL partition on the device with the `ftl_format` command. This layer hides the device−specific details of flash memory programming and make the card look like a simple block device. For example:

```
ftl_format −i /dev/mem0c0c
```

Note that this command accesses the card through the ``raw'' memory card interface. Once formatted, the card can be accessed as an ordinary block device via the `ftl_cs` driver. For example:

```
mke2fs /dev/ftl0c0
mount −t ext2 /dev/ftl0c0 /mnt
```

Device naming for FTL devices is tricky. Minor device numbers have three parts: the card number, the region number on that card, and optionally, the partition within that region. A region can either be treated as a single block device with no partition table (like a floppy), or it can be partitioned like a hard disk device. The ``ftl0c0'' device is card 0, common memory region 0, the entire region. The ``ftl0c0p1'' through ``ftl0c0p4'' devices are primary partitions 1 through 4 if the region has been partitioned.

Configuration options for FTL partitions can be given in `ftl.opts`, which is similar in structure to `memory.opts`. The device address passed to `ftl.opts` consists of three or four fields: the scheme, the socket number, the region number, and optionally, the partition number. Most flash cards have just one flash memory region, so the region number will generally always be zero.

Intel Series 100 flash cards use the first 128K flash block to store the cards' configuration information. To prevent accidental erasure of this information, `ftl_format` will automatically detect this and skip the first block when creating an FTL partition.

4.8 PCMCIA ATA/IDE card drives

ATA/IDE drive support is based on the regular kernel IDE driver. This includes SmartMedia and CompactFlash devices: these flash memory cards are set up so that they emulate an IDE interface. The PCMCIA−specific part of the driver is `ide_cs`. Be sure to use `cardctl` or `cardinfo` to shut down an ATA/IDE card before ejecting it, as the driver has not been made ``hot−swap−proof''.

The device addresses passed to `ide.opts` consist of either three or four fields: the current scheme, the socket number, the drive's serial number, and an optional partition number. The `ide_info` command can be used to obtain an IDE device's serial number. As with SCSI devices, `ide.opts` is first called for the entire device. If `ide.opts` returns a list of partitions in the `PARTS` variable, the script will then be called for each partition.

ATA/IDE fixed−disk device parameters

The following parameters can be specified in `ide.opts`:

DO_FSTAB

A boolean (y/n) setting: specifies if an entry should be added to `/etc/fstab` for this device.

DO_FSCK

A boolean (y/n) setting: specifies if the filesystem should be checked before being mounted, with ``fsck −Ta''.

DO_MOUNT

A boolean (y/n) setting: specifies if this device should be automatically mounted at card insertion time.

FSTYPE, OPTS, MOUNTPT

The filesystem type, mount options, and mount point to be used for the fstab entry and/or mounting the device.

NO_CHECK, NO_FUSER

Boolean (y/n) settings for card eject policy. If `NO_CHECK` is true, then ``cardctl eject'' will shut down a device even if it is busy. If `NO_FUSER` is true, then the script will not try to kill processes using an ejected device.

Here is an example `ide.opts` file to mount the first partition of any ATA/IDE card on `/mnt`.

```
case "$ADDRESS" in
*,*,*,1)
    DO_FSTAB="y" ; DO_FSCK="y" ; DO_MOUNT="y"
    FSTYPE="msdos"
    OPTS=""
    MOUNTPT="/mnt"
    ;;
*,*,*)
    PARTS="1"
    ;;
esac
```

Diagnosing problems with ATA/IDE adapters

- An IO port conflict may cause the IDE driver to misdetect the drive geometry and report ``INVALID GEOMETRY: 0 PHYSICAL HEADS?''. To fix, try excluding the selected IO port range in /etc/pcmcia/config.opts.
- Some IDE drives violate the PCMCIA specification by requiring more time to spin up than the maximum allowed card setup time. This may result in ``timed out during reset'' messages at card detect time. Adjust the unreset_delay and/or unreset_limit parameters for the pcmcia_core module to give a drive more time to spin up; see the pcmcia_core man page for parameter details. For example:

```
CORE_OPTS="unreset_delay=400"
```

- To use an ATA/IDE CD−ROM device, your kernel must be compiled with CONFIG_BLK_DEV_IDECD enabled. This will normally be the case for standard kernels, however it is something to be aware of if you compile a custom kernel.
- A common error when using IDE drives is try to mount the wrong device file. Generally, you want to mount a partition of the device, not the entire device (i.e., /dev/hde1, not /dev/hde).
- The Linux IDE driver may have trouble configuring certain removable−media drives if no media is present at insertion time. The IBM Portable DriveBay has this problem.
- Some kernels will report a pair of ``drive_cmd'' errors at insertion time. These errors can be ignored: they pop up when a removable IDE device does not accept the IDE ``door lock'' command.

4.9 Multifunction cards

A single interrupt can be shared by several drivers, such as the serial driver and an ethernet driver: in fact, the PCMCIA specification requires all card functions to share the same interrupt. Normally, all card functions are available without having to swap drivers. Any remotely recent Linux kernel (i.e., 1.3.72 or later) supports this kind of interrupt sharing.

Simultaneous use of two card functions is ``tricky'' and various hardware vendors have implemented interrupt sharing in their own incompatible (and sometimes proprietary) ways. The drivers for some cards (Ositech Jack of Diamonds, 3Com 3c562 and related cards, Linksys cards) properly support simultaneous access, but others (older Megahertz cards in particular) do not. If you have trouble using a card with both functions active, try using each function in isolation. That may require explicitly doing an ``ifconfig down'' to shut down a network interface and use a modem on the same card.

5.Advanced topics

5.1 Resource allocation for PCMCIA devices

In theory, it should not really matter which interrupt is allocated to which device, as long as two devices are not configured to use the same interrupt. In /etc/pcmcia/config.opts you'll find a place for excluding interrupts that are used by non−PCMCIA devices.

Similarly, there is no way to directly specify the I/O addresses for a card to use. The /etc/pcmcia/config.opts file allows you to specify ranges of ports available for use by any card, or to exclude ranges that conflict with other devices.

After modifying /etc/pcmcia/config.opts, you can reinitialize cardmgr with ``kill -HUP''.

The interrupt used to monitor card status changes is chosen by the low−level socket driver module (i82365 or tcic) before cardmgr parses /etc/pcmcia/config, so it is not affected by changes to this file. To set this interrupt, use the cs_irq= option when the socket driver is loaded, by setting the PCIC_OPTS variable in /etc/rc.d/rc.pcmcia.

All the client card drivers have a parameter called irq_list for specifying which interrupts they may try to allocate. These driver options should be set in your /etc/pcmcia/config file. For example:

```
device "serial_cs"
  module "serial_cs" opts "irq_list=8,12"
  ...
```

would specify that the serial driver should only use irq 8 or irq 12. Regardless of irq_list settings, Card Services will never allocate an interrupt that is already in use by another device, or an interrupt that is excluded in the config file.

5.2 PCI interrupt configuration problems and solutions

An overview of PCI interrupt routing issues

Each PCI slot has four PCI interrupt pins, INTA through INTD. Single function devices will only use the INTA pin; multifunction devices may use multiple INT pins. On the processor side, on x86 single processor systems, incoming hardware interrupts are directed to interrupt requests (irq's) numbered 0..15. The PCI interrupt router, usually part of the PCI−to−ISA host bridge, determines how incoming PCI interrupts are mapped to CPU irq numbers. Most modern bridge chips have several PCI interrupt inputs, known as PIRQ1, PIRQ2, etc, each of which can be routed to any CPU irq number. So we might have something like:

```
PCI slot 1 INTA --> router PIRQ1 --> CPU irq 9
PCI slot 1 INTB --> router PIRQ2 --> CPU irq 10

PCI slot 2 INTA --> router PIRQ2 --> CPU irq 10
PCI slot 2 INTB --> router PIRQ1 --> CPU irq 9
```

Multiple INT pins are often connected to the same PIRQ pin. Usually, the connections from INT pins to PIRQ pins are arranged to spread installed devices out as much as possible, to give the OS the most flexibility for choosing how interrupts are shared. The mapping from bridge PIRQ pins to CPU irq numbers can be obtained by reading registers in the interrupt router. The mapping from INT pins to the router's PIRQ pins, however, depends on how the board designer decided to connect things up, and cannot be directly determined by driver software.

For most PCI devices, the OS does not need to understand the interrupt router details. Each PCI device has a configuration register, the PCI Interrupt Line Register, that the BIOS initializes with the appropriate CPU irq number for that device. Unfortunately, the BIOS generally will not configure PCI interrupts for CardBus bridge devices.

The PCI BIOS's Interrupt Routing Table is a data structure that contains information about the mapping from PCI INT pins to the PIRQ pins on the PCI interrupt router. The routing information in the table is stored in a somewhat unhelpful form, however. For each device's INT pins, the table specifies a ``link value''. All interrupts with the same link value are wired to the same PIRQ pin; however, the meaning of the link values is defined by the chipset vendor.

Several tools are available for examining PCI interrupt routing information:

lspci*, */proc/pci

These will show you resource information (including interrupt assignments, where they are known) for all your PCI devices.

dump_pirq

This is in the `debug-tools` directory of the PCMCIA source distribution. It dumps the contents of your PCI interrupt routing table, if available. It also scans for known interrupt routers and dumps their current interrupt steering settings.

Several PCMCIA module parameters affect PCI interrupt routing:

pcmcia_core* module: *cb_pci_irq=n

This option specifies one interrupt number to be used to program the PCI interrupt router for all CardBus sockets that do not already have an interrupt assignment. It only has any effect on systems that have a PCI irq routing table, and a known interrupt router.

i82365* module: *irq_mode=n

Most CardBus bridges offer several methods for delivering interrupts to the host. The i82365 module by default assumes that a bridge can deliver both PCI and ISA interrupts, since this is normal for laptops. A setting of ``irq_mode=0'' can be used to force a bridge to use only PCI interrupts. See the man page for the i82365 module for a description of what other values mean for different bridge types.

i82365* module: *irq_list=n,n,...

This parameter lists which ISA interrupt(s) can be used for PCMCIA. If no ISA interrupts are available, specify ``irq_list=0''. Note that ``irq_mode=0'' implies ``irq_list=0''.

i82365* module: *pci_irq_list=n,n,...

This option specifies a list of PCI interrupt numbers to use for CardBus sockets. It differs from cb_pci_irq, because it does not actually program the PCI interrupt router; it can be used when you know the PCI interrupts are already set up a certain way, even if you do not know how the router works.

If you are having problems that you think may be related to PCI interrupt configuration, you should first verify that you have a reasonably current PCMCIA driver package. Also carefully look at the startup messages when the PCMCIA kernel modules are loaded. You should see something like:

```
Linux PCMCIA Card Services 3.1.18
  kernel build: 2.2.14-5.0 #1 Tue May 9 10:44:24 PDT 2000
  options:  [pci] [cardbus] [apm] [pnp]
PCI routing table version 1.0 at 0xfdf30
Intel PCIC probe:
  TI 1125 rev 02 PCI-to-CardBus at slot 00:07, mem 0x20000000
    host opts [0]: [ring] [serial pci & irq] [pci irq 11] ...
    host opts [0]: [ring] [serial pci & irq] [pci irq 11] ...
  ISA irqs (scanned) = 3,4,7 PCI status changes
```

The ``PCI routing table'' message indicates that a valid routing table was found. The ``host opts'' lines indicate the interrupt delivery mode and whether or not a PCI interrupt could be determined for each socket. And the final line indicates the results of the scan for available interrupts.

CardBus bridge is not detected by the PCI BIOS

Symptoms:

- Intel PCIC probe: not found.
- The bridge does not show up in `lspci` or in `/proc/pci`.

The Lucent/SCM PCI−to−CardBus adapters seem to confuse the PCI BIOS on some older systems. Lucent says that this card is only supported on systems that have a BIOS that supports the PCI 2.2 specification, or are PC99 compliant. Some older systems will not detect the Lucent card at all, and if the system can't detect it, the Linux drivers cannot use it. The only possible resolutions are a BIOS upgrade, or using a different motherboard or CardBus adapter.

PCI interrupt delivery problems

Symptoms:

- Cards seem to be configured correctly, but do not work.
- `/proc/interrupts` shows a count of 0 for interrupts assigned to PCMCIA drivers.

CardBus bridges usually support two types of interrupts, PCI and ISA. Partly for historical reasons, it has become conventional to use PCI interrupts for signaling card insertion and removal events, and for CardBus card interrupts; and ISA interrupts for 16−bit cards. Since version 3.1.9, this is the scheme that the Linux PCMCIA system will use by default. Most CardBus bridges support multiple methods for delivering interrupts to the host CPU. Methods include ``parallel'' interrupts, where each supported irq has a dedicated pin on the bridge; various serial interrupt protocols, where one or two pins are used to communicate with an interrupt controller; and hybrids, where PCI interrupts might be signalled using dedicated pins, while ISA interrupts are delivered via a serial controller.

In general, it is the responsibility of the BIOS to program a bridge for the appropriate interrupt delivery method. However, there are systems that do this incorrectly, and in some cases, there is no way for software to safely detect the correct delivery method. The `i82365` module reports the bridge mode at startup time, and has a parameter, `irq_mode`, that can be used to reconfigure it. Not all bridges support this parameter, and the meaning of `irq_mode` depends on the bridge type. See the `i82365` man page for a description of what values are supported by your bridge. In some cases, a bridge may function correctly in more than one interrupt mode.

Most PCMCIA card readers that fit in a PCI bus slot only provide PCI interrupt routing. The Linux drivers assume that all bridges have ISA interrupt capability, since that is generally correct on laptops. With a card reader, it will generally be necessary to use the `irq_mode` parameter to specify a ``PCI only'' interrupt delivery mode; the value of the parameter depends on the bridge type, so check the `i82365` man page. A few PCI card readers require an `irq_mode` that permits ISA interrupts, but those interrupts are not actually connected; in that case, use ``irq_list=0''.

Check the system log and verify that the CardBus bridge has a PCI interrupt assignment. If it does not, then resolve that problem first, then return here if the symptoms persist. Next, experiment with different values for the `irq_mode` parameter.

No PCI interrupt assignment; valid routing table

Symptoms:

- The Intel PCIC probe reports ``no pci irq'' for each socket.
- There is a routing table, and the router type is supported.

When a routing table is present, the `pcmcia_core` module will try to automatically configure the PCI interrupt router, but only does so when it has a safe and unambiguous choice for what PCI interrupt to use. If there are several valid choices, then you must use the ``cb_pci_irq=...'' option to specify which interrupt to assign. Your best bet is to pick the most lightly used interrupt that is already assigned to another PCI device.

Moving the card to another slot sometimes offers a quick solution. If that slot shares its interrupt with an already−configured device, then the PCMCIA drivers will have no trouble figuring out the assignment.

No PCI interrupt assignment; unknown interrupt router

Symptoms:

- The Intel PCIC probe reports ``no pci irq'' for each socket.
- There is a routing table, but the router is an unknown type.

Adding support for a new interrupt router is tricky but not a big job. First determine, from a datasheet, how your interrupt router steers PCI interrupts. Then, see if you can guess the meaning of the link values from the output of `dump_pirq`. Usually this is reasonably obvious. Most routers have four PIRQ pins, and the link values might be something like 1,2,3,4, or 0x10,0x18,0x20,0x28, or 0x60,0x61,0x62,0x63. The values are usually chosen so that they can be easily converted to the location of the appropriate interrupt steering register. Finally, add small functions to `modules/pci_fixup.c` to get/set the interrupt steering information for this router, using the other routers as examples.

No PCI interrupt assignment; no routing table

Symptoms:

- The Intel PCIC probe reports ``no pci irq'' for each socket.
- No interrupt routing table is found.

Without an interrupt routing table, we cannot tell how interrupts from the CardBus bridge are directed to

CPU irq numbers. All hope is not lost: you may be able to guess the PCI interrupt assignment and use the ``pci_irq_list=...'' option to pass this information to the i82365 module. Good guesses might include the interrupt(s) assigned to other PCI devices, the interrupt(s) used under Windows, or any other interrupts that are unaccounted for.

You may also want to experiment with putting the adapter in different PCI slots, for each pci_irq_list you try. You are trying to find a slot that shares its interrupt with an already−configured device, and might need to try several slots to find one.

5.3 How can I have separate device setups for home and work?

This is fairly easy using ``scheme'' support. Use two configuration schemes, called ``home'' and ``work''. Here is an example of a network.opts script with scheme−specific settings:

```
case "$ADDRESS" in
work,*,*,*)
    # definitions for network card in work scheme
    ...
    ;;
home,*,*,*|default,*,*,*)
    # definitions for network card in home scheme
    ...
    ;;
esac
```

The first part of a device address is always the configuration scheme. In this example, the second ``case'' clause will select for both the ``home'' and ``default'' schemes. So, if the scheme is unset for any reason, it will default to the ``home'' setup.

Now, to select between the two sets of settings, run either:

```
cardctl scheme home
```

or

```
cardctl scheme work
```

The cardctl command does the equivalent of shutting down all your cards and restarting them. The command can be safely executed whether or not the PCMCIA system is loaded, but the command may fail if you are using other PCMCIA devices at the time (even if their configurations are not explicitly dependant on the scheme setting).

To find out the current scheme setting, run:

```
cardctl scheme
```

By default, the scheme setting is persistent across boots. This can have undesirable effects if networking is initialized for the wrong environment. Optionally, you can set the initial scheme value with the SCHEME startup option (see Startup options for details). It is also possible to set the scheme from the lilo boot prompt. Since lilo passes unrecognized options to init as environment variables, a value for SCHEME (or any other PCMCIA startup option) at the boot prompt will be propagated into the PCMCIA startup script.

To save even more keystrokes, schemes can be specified in lilo's configuration file. For instance, you could have:

```
root = /dev/hda1
read-only
image = /boot/vmlinuz
  label  = home
  append = "SCHEME=home"
image = /boot/vmlinuz
  label  = work
  append = "SCHEME=work"
```

Typing ``home'' or ``work'' at the boot prompt would then boot into the appropriate scheme.

5.4 Booting from a PCMCIA device

Having the root filesystem on a PCMCIA device is tricky because the Linux PCMCIA system is not designed to be linked into the kernel. Its core components, the loadable kernel modules and the user mode cardmgr daemon, depend on an already running system. The kernel's ``initrd'' facility works around this requirement by allowing Linux to boot using a temporary ram disk as a minimal root image, load drivers, and then re−mount a different root filesystem. The temporary root can configure PCMCIA devices and then re−mount a PCMCIA device as root.

The initrd image absolutely must reside on a bootable device: this generally cannot be put on a PCMCIA device. This is a BIOS limitation, not a kernel limitation. It is useful here to distinguish between ``boot−able'' devices (i.e., devices that can be booted), and ``root−able'' devices (i.e., devices that can be mounted as root). ``Boot−able'' devices are determined by the BIOS, and are generally limited to internal floppy and hard disk drives. ``Root−able'' devices are any block devices that the kernel supports once it has been loaded. The initrd facility makes more devices ``root−able'', not ``boot−able''.

Some Linux distributions will allow installation to a device connected to a PCMCIA SCSI adapter, as an unintended side−effect of their support for installs from PCMCIA SCSI CD−ROM devices. However, at present, no Linux installation tools support configuring an appropriate ``initrd'' to boot Linux with a

PCMCIA root filesystem. Setting up a system with a PCMCIA root thus requires that you use another Linux system to create the ``initrd'' image. If another Linux system is not available, another option would be to temporarily install a minimal Linux setup on a non−PCMCIA drive, create an initrd image, and then reinstall to the PCMCIA target.

The Linux Bootdisk−HOWTO has some general information about setting up boot disks but nothing specific to initrd. The main initrd document is included with recent kernel source code distributions, in `linux/Documentation/initrd.txt`. Before beginning, you should read this document. A familiarity with `lilo` is also helpful. Using initrd also requires that you have a kernel compiled with `CONFIG_BLK_DEV_RAM` and `CONFIG_BLK_DEV_INITRD` enabled.

This is an advanced configuration technique, and requires a high level of familiarity with Linux and the PCMCIA system. Be sure to read all the relevant documentation before starting. The following cookbook instructions should work, but deviations from the examples will quickly put you in uncharted and ``unsupported'' territory, and you will be on your own.

This method absolutely requires that you use a PCMCIA driver release of 2.9.5 or later. Older PCMCIA packages or individual components will not work in the initrd context. Do not mix components from different releases.

The pcinitrd helper script

The `pcinitrd` script creates a basic initrd image for booting with a PCMCIA root partition. The image includes a minimal directory heirarchy, a handful of device files, a few binaries, shared libraries, and a set of PCMCIA driver modules. When invoking `pcinitrd`, you specify the driver modules that you want to be included in the image. The core PCMCIA components, `pcmcia_core` and `ds`, are automatically included.

As an example, say that your laptop uses an i82365−compatible host controller, and you want to boot Linux with the root filesystem on a hard drive attached to an Adaptec SlimSCSI adapter. You could create an appropriate initrd image with:

```
pcinitrd -v initrd pcmcia/i82365.o pcmcia/aha152x_cs.o
```

To customize the initrd startup sequence, you could mount the image using the ``loopback'' device with a command like:

```
mount -o loop -t ext2 initrd /mnt
```

and then edit the `linuxrc` script. The configuration files will be installed under `/etc` in the image, and can also be customized. See the man page for `pcinitrd` for more information.

Creating an initrd boot floppy

After creating an image with `pcinitrd`, you can create a boot floppy by copying the kernel, the compressed initrd image, and a few support files for `lilo` to a clean floppy. In the following example, we assume that the desired PCMCIA root device is `/dev/sda1`:

```
mke2fs /dev/fd0
mount /dev/fd0 /mnt
mkdir /mnt/etc /mnt/boot /mnt/dev
cp -a /dev/fd0 /dev/sda1 /mnt/dev
cp [kernel-image] /mnt/vmlinuz
cp /boot/boot.b /mnt/boot/boot.b
gzip < [initrd-image] > /mnt/initrd
```

Create `/mnt/etc/lilo.conf` with the contents:

```
boot=/dev/fd0
compact
image=/vmlinuz
    label=linux
    initrd=/initrd
    read-only
    root=/dev/sda1
```

Finally, invoke lilo with:

```
lilo -r /mnt
```

When `lilo` is invoked with `-r`, it performs all actions relative to the specified alternate root directory. The reason for creating the device files under `/mnt/dev` was that `lilo` will not be able to use the files in `/dev` when it is running in this alternate-root mode.

Installing an initrd image on a non-Linux drive

One common use of the initrd facility would be on systems where the internal hard drive is dedicated to another operating system. The Linux kernel and initrd image can be placed in a non-Linux partition, and `lilo` or `LOADLIN` can be set up to boot Linux from these images.

Assuming that you have a kernel has been configured for the appropriate root device, and an initrd image created on another system, the easiest way to get started is to boot Linux using `LOADLIN`, as:

```
LOADLIN <kernel> initrd=<initrd-image>
```

Once you can boot Linux on your target machine, you could then install `lilo` to allow booting Linux directly. For example, say that `/dev/hda1` is the non−Linux target partition and `/mnt` can be used as a mount point. First, create a subdirectory on the target for the Linux files:

```
mount /dev/hda1 /mnt
mkdir /mnt/linux
cp [kernel-image] /mnt/linux/vmlinuz
cp [initrd-image] /mnt/linux/initrd
```

In this example, say that `/dev/sda1` is the desired Linux root partition, a SCSI hard drive mounted via a PCMCIA SCSI adapter. To install `lilo`, create a `lilo.conf` file with the contents:

```
boot=/dev/hda
map=/mnt/linux/map
compact
image=/mnt/linux/vmlinuz
        label=linux
        root=/dev/sda1
        initrd=/mnt/linux/initrd
        read-only
other=/dev/hda1
        table=/dev/hda
        label=windows
```

The `boot=` line says to install the boot loader in the master boot record of the specified device. The `root=` line identifies the desired root filesystem to be used after loading the initrd image, and may be unnecessary if the kernel image is already configured this way. The `other=` section is used to describe the other operating system installed on `/dev/hda1`.

To install `lilo` in this case, use:

```
lilo -C lilo.conf
```

Note that in this case, the `lilo.conf` file uses absolute paths that include `/mnt`. I did this in the example because the target filesystem may not support the creation of Linux device files for the `boot=` and `root=` options.

6.Dealing with unsupported cards

6.1 Configuring unrecognized cards

Assuming that your card is supported by an existing driver, all that needs to be done is to add an entry to `/etc/pcmcia/config` to tell `cardmgr` how to identify the card, and which driver(s) need to be linked up to this card. Check the man page for `pcmcia` for more information about the config file format. If you insert an unknown card, `cardmgr` will normally record some identification information in the system log that can be used to construct the config entry. This information can also be displayed with the ``` ``cardctl ident'' ``` command.

Here is an example of how cardmgr will report an unsupported card in `/usr/adm/messages`.

```
cardmgr[460]: unsupported card in socket 1
cardmgr[460]: product info: "MEGAHERTZ", "XJ2288", "V.34 PCMCIA MODEM"
cardmgr[460]: manfid: 0x0101, 0x1234  function: 2 (serial)
```

The corresponding entry in `/etc/pcmcia/config` would be:

```
card "Megahertz XJ2288 V.34 Fax Modem"
  version "MEGAHERTZ", "XJ2288", "V.34 PCMCIA MODEM"
  bind "serial_cs"
```

or using the more compact product ID codes:

```
card "Megahertz XJ2288 V.34 Fax Modem"
  manfid 0x0101, 0x1234
  bind "serial_cs"
```

You can use ``*'' to match strings that don't need to match exactly, like version numbers. When making new config entries, be careful to copy the strings exactly, preserving case and blank spaces. Also be sure that the config entry has the same number of strings as are reported in the log file.

Beware that you can specify just about any driver for a card, but if you're just shooting in the dark, there is not much reason to expect this to be productive. You may get lucky and find that your card is supported by an existing driver. However, the most likely outcome is that the driver won't work, and may have unfortunate side effects like locking up your system. Unlike most ordinary device drivers, which probe for an appropriate card, the probe for a PCMCIA device is done by `cardmgr`, and the driver itself may not do much validation before attempting to communicate with the device.

After editing /etc/pcmcia/config, you can signal cardmgr to reload the file with:

```
kill -HUP `cat /var/run/cardmgr.pid`
```

If you do set up an entry for a new card, please send me a copy so that I can include it in the standard config file.

6.2 Adding support for an NE2000−compatible ethernet card

Before you begin: this procedure will only work for simple ethernet cards. Multifunction cards (i.e., ethernet/modem combo cards) have an extra layer of complexity regarding how the two functions are integrated, and generally cannot be supported without obtaining some configuration information from the card vendor. Using the following procedure for a multifunction card will not be productive.

First, see if the card is already recognized by cardmgr. Some cards not listed in SUPPORTED.CARDS are actually OEM versions of cards that are supported. If you find a card like this, let me know so I can add it to the list.

If your card is not recognized, follow the instructions in the Configuring unrecognized cards section to create a config entry for your card, and bind the card to the pcnet_cs driver. Restart cardmgr to use the updated config file.

If the pcnet_cs driver says that it is unable to determine your card's hardware ethernet address, then edit your new config entry to bind the card to the memory card driver, memory_cs. Restart cardmgr to use the new updated config file. You will need to know your card's hardware ethernet address. This address is a series of six two−digit hex numbers, often printed on the card itself. If it is not printed on the card, you may be able to use a DOS driver to display the address. In any case, once you know it, run:

```
dd if=/dev/mem0a count=20 | od −Ax −t x1
```

and search the output for your address. Only the even bytes are defined, so ignore the odd bytes in the dump. Record the hex offset of the first byte of the address. Now, edit clients/pcnet_cs.c and find the hw_info structure. You'll need to create a new entry for your card. The first field is the memory offset. The next three fields are the first three bytes of the hardware address. The final field contains some flags for specific card features; to start, try setting it to 0.

After editing pcnet_cs.c, compile and install the new module. Edit /etc/pcmcia/config again, and change the card binding from memory_cs to pcnet_cs. Follow the instructions for reloading the config file, and you should be all set. Please send me copies of your new hw_info and config entries.

If you can't find your card's hardware address in the hex dump, as a method of last resort, it is possible to ``hard−wire'' the address when the pcnet_cs module is initialized. Edit /etc/pcmcia/config.opts and add a hw_addr= option, like so:

```
module "pcnet_cs" opts "hw_addr=0x00,0x80,0xc8,0x01,0x02,0x03"
```

Substitute your own card's hardware address in the appropriate spot, of course. Beware that if you've gotten this far, it is very unlikely that your card is genuinely NE2000 compatible. In fact, I'm not sure if there are *any* cards that are not handled by one of the first two methods.

6.3 PCMCIA floppy interface cards

The PCMCIA floppy interface used in the Compaq Aero and a few other laptops is not yet supported by this package. The snag in supporting the Aero floppy is that the Aero seems to use a customized PCMCIA controller to support DMA to the floppy. Without knowing exactly how this is done, there isn't any way to implement support under Linux.

If the floppy adapter card is present when an Aero is booted, the Aero BIOS will configure the card, and Linux will identify it as a normal floppy drive. When the Linux PCMCIA drivers are loaded, they will notice that the card is already configured and attached to a Linux driver, and this socket will be left alone. So, the drive can be used if it is present at boot time, but the card is not hot swappable.

7.Debugging tips and programming information

7.1 Submitting useful bug reports

The best way to submit bug reports is to use the HyperNews message lists on the Linux PCMCIA information site. That way, other people can see current problems (and fixes or workarounds, if available). Here are some things that should be included in all bug reports:

- Your system brand and model.
- What PCMCIA card(s) you are using.
- Your Linux kernel version (i.e., ``uname -rv''), and PCMCIA driver version (i.e., ``cardctl -V'').
- Any changes you have made to the startup files in `/etc/pcmcia`, or to the PCMCIA startup script.
- All PCMCIA−related messages in your system log file. That includes startup messages, and messages generated when your cards are configured.

All the PCMCIA modules and the `cardmgr` daemon send status messages to the system log. This will usually be something like `/var/log/messages` or `/usr/adm/messages`. This file should be the first place to look when tracking down a problem. When submitting a bug report, always include the relevant contents of this file. If you are having trouble finding your system messages, check `/etc/syslog.conf` to see how different classes of messages are handled.

Before submitting a bug report, please check to make sure that you are using an up−to−date copy of the driver package. While it is somewhat gratifying to read bug reports for things I've already fixed, it isn't a particularly constructive use of my time.

If you do not have web access, bug reports can be sent to me at `dahinds@users.sourceforge.net`. However, I prefer that bug reports be posted to the PCMCIA web site, so that they can be seen by others.

7.2 Interpreting kernel trap reports

If your problem involves a kernel fault, the register dump from the fault is only useful if you can translate the fault address, EIP, to something meaningful. Recent versions of `klogd` attempt to translate fault addresses based on the current kernel symbol map, but this may not work if the fault is in a module, or if the problem is severe enough that `klogd` cannot finish writing the fault information to the system log.

If a fault is in the main kernel, the fault address can be looked up in the `System.map` file. This may be installed in `/System.map` or `/boot/System.map`. If a fault is in a module, the `nm` command gives the same information, however, the fault address has to be adjusted based on the module's load address. Let's say that you have the following kernel fault:

```
Unable to handle kernel NULL pointer dereference
current->tss.cr3 = 014c9000, %cr3 = 014c9000
*pde = 00000000
Oops: 0002
CPU:    0
EIP:    0010:[<c2026081>]
EFLAGS: 00010282
```

The fault address is 0xc2026081. Looking at `System.map`, we see that this is past the end of the kernel, i.e., is in a kernel module. To determine which module, check the output of ``` ``ksyms -m | sort'' ```:

```
Address   Symbol                             Defined by
c200d000  (35k)                              [pcmcia_core]
c200d10c  register_ss_entry                  [pcmcia_core]
c200d230  unregister_ss_entry                [pcmcia_core]
          ...
c2026000  (9k)                               [3c574_cs]
c202a000  (4k)                               [serial_cs]
```

So, 0xc2026081 is in the `3c574_cs` module, and is at an offset of 0x0081 from the start of the module. We cannot look up this offset in `3c574_cs.o` yet: when the kernel loads a module, it inserts a header at the module load address, so the real start of the module is offset from the address shown in `ksyms`. The size of the header varies with kernel version: to find out the size for your kernel, check a module that exports symbols (like `pcmcia_core` above), and compare a symbol address with `nm` output for that symbol. In this example, `register_ss_entry` is loaded at an offset of 0xc200d10c – 0xc200d000 = 0x010c, while ``` ``nm pcmcia_core.o'' ``` shows the offset as 0x00c0, so the header size is 0x010c – 0x00c0 = 0x004c bytes.

Back to `3c574_cs`, our fault offset is 0x0081, and subtracting the 0x004c header, the real module offset is

0x0035. Now looking at ``nm 3c574_cs.o | sort'', we see:

```
0000002c d if_names
0000002c t tc574_attach
00000040 d mii_preamble_required
00000041 d dev_info
```

So, the fault is located in `tc574_attach()`.

In this example, the fault did not cause a total system lockup, so `ksyms` could be executed after the fault happened. In other cases, you may have to infer the module load addresses indirectly. The same sequence of events will normally load modules in the same order and at the same addresses. If a fault happens when a certain card is inserted, get the `ksyms` output before inserting the card, or with a different card inserted. You can also manually load the card's driver modules with `insmod` and run `ksyms` before inserting the card.

For more background, see ``man insmod'', ``man ksyms'', and ``man klogd''. In the kernel source tree, `Documentation/oops-tracing.txt` is also relevant. Here are a few more kernel debugging hints:

- Depending on the fault, it may also be useful to translate addresses in the ``Call Trace'', using the same procedure as for the main fault address.
- To diagnose a silent lock−up, try to provoke the problem with X disabled, since kernel messages sent to the text console will not be visible under X.
- If you kill `klogd`, most kernel messages will be echoed directly on the text console, which may be helpful if the problem prevents `klogd` from writing to the system log.
- To cause all kernel messages to be sent to the console, for 2.1 kernels, if `/proc/sys/kernel/printk` exists, do:

  ```
  echo 8 > /proc/sys/kernel/printk
  ```

- The key combination <RightAlt><ScrLk> prints a register dump on the text console. This may work even if the system is otherwise completely unresponsive, and the EIP address can be interpreted as for a kernel fault.
- For 2.1 kernels configured with `CONFIG_MAGIC_SYSRQ` enabled, various emergency functions are available via special <Alt><SysRq> key combinations, documented in `Documentation/sysrq.txt` in the kernel source tree.

7.3 Low level PCMCIA debugging aids

The PCMCIA modules contain a lot of conditionally−compiled debugging code. Most of this code is under control of the `PCMCIA_DEBUG` preprocessor define. If this is undefined, debugging code will not be compiled. If set to 0, the code is compiled but inactive. Larger numbers specify increasing levels of verbosity. Each module built with `PCMCIA_DEBUG` defined will have an integer parameter, `pc_debug`, that controls the verbosity of its output. This can be adjusted when the module is loaded, so output can be controlled on a per−module basis without recompiling.

Your default configuration for `syslogd` may discard kernel debugging messages. To ensure that they are recorded, edit `/etc/syslog.conf` to ensure that ``kern.debug'' messages are recorded somewhere. See ``man syslog.conf'' for details.

There are a few register−level debugging tools in the `debug_tools/` subdirectory of the PCMCIA distribution. The `dump_tcic` and `dump_i365` utilities generate register dumps for ISA−to−PCMCIA controllers. In 3.1.15 and later releases, `dump_i365` is replaced by `dump_exca`, which is similar but also works for PCI−to−CardBus bridges. Also new in 3.1.15 for CardBus bridges is the `dump_cardbus` tool, which interprets the CardBus−specific registers. These are all most useful if you have access to a datasheet for the corresponding controller chip. The `dump_cis` utility (`dump_tuples` in pre−3.0.2 distributions) lists the contents of a card's CIS (Card Information Structure), and decodes most of the important bits. And the `dump_cisreg` utility displays a card's local configuration registers.

The `memory_cs` memory card driver is also sometimes useful for debugging problems with 16−bit PC Cards. It can be bound to any card, and does not interfere with other drivers. It can be used to directly access any card's attribute memory or common memory. Similarly for CardBus cards, the `memory_cb` driver can be bound to any 32−bit card, to give direct access to that card's address spaces. See the man pages for more information.

7.4 /proc/bus/pccard

Starting with 2.1.103 kernels, the PCMCIA package will create a tree of status information under `/proc/bus/pccard`. Much of the information can only be interpreted using the data sheets for the PCMCIA host controller. Its contents may depend on how the drivers were configured, but may include all or some of the following:

/proc/bus/pccard/{irq,ioport,memory}

If present, these files contain resource allocation information to supplement the normal kernel resource tables. Recent versions of the PCMCIA system may obtain additional resource information from the Plug and Play BIOS if configured to do so.

/proc/bus/pccard/drivers

In recent releases, this lists all currently loaded PCMCIA client drivers. Unlike `/proc/modules`, it also lists drivers that may be statically linked into the kernel.

/proc/bus/pccard/*/info

For each socket, describes that socket's host controller and its capabilities.

/proc/bus/pccard/*/exca

This contains a dump of a controller's ``ExCA'' Intel i82365sl−compatible register set.

/proc/bus/pccard/*/{pci,cardbus}

For CardBus bridges, a dump of the bridge's PCI configuration space, and a dump of the bridge's CardBus configuration registers.

7.5 Writing Card Services drivers for new cards

The Linux PCMCIA Programmer's Guide is the best documentation for the client driver interface. The latest version is always available from `projects.sourceforge.net` in `/pub/pcmcia-cs/doc`, or on the web at http://pcmcia–cs.sourceforge.net.

For devices that are close relatives of normal ISA devices, you will probably be able to use parts of existing Linux drivers. In some cases, the biggest stumbling block will be modifying an existing driver so that it can handle adding and removing devices after boot time. Of the current drivers, the memory card driver is the only ``self–contained'' driver that does not depend on other parts of the Linux kernel to do most of the dirty work.

In many cases, the largest barrier to supporting a new card type is obtaining technical information from the manufacturer. It may be difficult to figure out who to ask, or to explain exactly what information is needed. However, with a few exceptions, it is very difficult if not impossible to implement a driver for a card without technical information from the manufacturer.

I have written a dummy driver with lots of comments that explains a lot of how a driver communicates with Card Services; you will find this in the PCMCIA source distribution in `clients/dummy_cs.c`.

7.6 Guidelines for PCMCIA client driver authors

I have decided that it is not really feasible for me to distribute all PCMCIA client drivers as part of the PCMCIA package. Each new driver makes the main package incrementally harder to maintain, and including a driver inevitably transfers some of the maintenance work from the driver author to me. Instead, I will decide on a case by case basis whether or not to include contributed drivers, based on user demand as well as maintainability. For drivers not included in the core package, I suggest that driver authors adopt the following scheme for packaging their drivers for distribution.

Driver files should be arranged in the same directory scheme used in the PCMCIA source distribution, so that the driver can be unpacked on top of a complete PCMCIA source tree. A driver should include source files (in `./modules/`), a man page (in `./man/`), and configuration files (in `./etc/`). The top level directory should also include a README file.

The top–level directory should include a makefile, set up so that ``make -f ... all'' and ``make -f ... install'' compile the driver and install all appropriate files. If this makefile is given an extension of `.mk`, then it will automatically be invoked by the top–level `Makefile` for the `all` and `install` targets. Here is an example of how such a makefile could be constructed:

```
# Sample Makefile for contributed client driver
FILES = sample_cs.mk README.sample_cs \
        modules/sample_cs.c modules/sample_cs.h \
```

```
        etc/sample.conf etc/sample etc/sample.opts \
        man/sample_cs.4
all:
        $(MAKE) -C modules MODULES=sample_cs.o
install:
        $(MAKE) -C modules install-modules MODULES=sample_cs.o
        $(MAKE) -C etc install-clients CLIENTS=sample
        $(MAKE) -C man install-man4 MAN4=sample_cs.4
dist:
        tar czvf sample_cs.tar.gz $(FILES)
```

This makefile uses install targets defined in 2.9.10 and later versions of the PCMCIA package. This makefile also includes a ``dist'' target for the convenience of the driver author. You would probably want to add a version number to the final package filename (for example, `sample_cs-1.5.tar.gz`). A complete distribution could look like:

```
sample_cs.mk
README.sample_cs
modules/sample_cs.c
modules/sample_cs.h
etc/sample.conf
etc/sample
etc/sample.opts
man/sample_cs.4
```

With this arrangement, when the contributed driver is unpacked, it becomes essentially part of the PCMCIA source tree. It can make use of the PCMCIA header files, as well as the machinery for checking the user's system configuration, and automatic dependency checking, just like a ``normal'' client driver.

In this example, `etc/sample` and `etc/sample.opts` would be the new driver's configuration scripts (if needed), and `etc/sample.conf` would contain any additions to the PCMCIA card configuration file. Starting with the 3.1.6 release, `cardmgr` will automatically process any `*.conf` files installed in `/etc/pcmcia`, so installation of contributed drivers should no longer require hand editing configuration files.

I will accept client drivers prepared according to this specification and place them in the `/pub/pcmcia-cs/contrib` directory on `projects.sourceforge.net`. The README in this directory will describe how to unpack a contributed driver.

The client driver interface has not changed much over time, and has almost always preserved backwards compatibility. A client driver will not normally need to be updated for minor revisions in the main package. I will try to notify authors of contributed drivers of changes that require updates to their drivers.

7.7 Guidelines for Linux distribution maintainers

If your distribution has system configuration tools that you would like to be PCMCIA−aware, please use the `*.opts` files in `/etc/pcmcia` for your ``hooks.'' These files will not be modified if a user compiles and installs a new release of the PCMCIA package. If you modify the main configuration scripts, then a fresh install will silently overwrite your custom scripts and break the connection with your configuration tools. Contact me if you are not sure how to write an appropriate option script, or if you need additional capabilities.

It is helpful for users (and for me) if you can document how your distribution deviates from the PCMCIA package as described in this document. In particular, please document changes to the startup script and configuration scripts. If you send me the appropriate information, I will include it in the Notes about specific Linux distributions.

When building PCMCIA for distribution, consider including contributed drivers that are not part of the main PCMCIA package. For reasons of maintainability, I am trying to limit the core package size, by only adding new drivers if I think they are of particularly broad interest. Other drivers will be distributed separately, as described in the previous section. The split between integral and separate drivers is somewhat arbitrary and partly historical, and should not imply a difference in quality.
