Linux PCMCIA HOWTO

David Hinds, dhinds@hyper.stanford.edu v1.105, 9 October 1997

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 hyper.stanford.edu in /pub/pcmcia/doc. An HTML version is at
http://hyper.stanford.edu/HyperNews/get/pcmcia/home.html

Contents

1 General information and hardware requirements **3**

1.1 Introduction . . . 3

1.2 Copyright notice and disclaimer . . . 3

1.3 What is the latest version, and where can I get it? . . . 3

1.4 What systems are supported? . . . 4

1.5 What PCMCIA cards are supported? . . . 4

1.6 When will my new card be supported? . . . 4

1.7 Mailing lists . . . 5

2 Compilation, installation, and configuration **5**

2.1 Prerequisites and kernel setup . . . 5

2.2 Installation . . . 6

2.3 Post-installation for systems using BSD init scripts . . . 8

2.4 Post-installation for systems using System V init scripts . . . 8

2.5 Site-specific configuration options . . . 8

2.6 Problems loading kernel modules . . . 10

2.7 Problems with the card status change interrupt . . . 11

2.8 Card identification problems . . . 11

2.9 Why don't you distribute PCMCIA binaries? . . . 12

2.10 Why is the PCMCIA package so darned big? . . . 12

3 Usage and features **13**

3.1 Tools for monitoring PCMCIA devices . . . 13

3.2 Overview of the PCMCIA configuration scripts . . . 14

3.3 PCMCIA network adapters . . . 14

3.3.1 Transceiver selection . . . 16
3.3.2 Comments about specific cards . . . 16
3.3.3 Diagnosing problems with network adapters . . . 17
3.4 PCMCIA serial and modem devices . . . 17
3.4.1 Diagnosing problems with serial devices . . . 18
3.5 PCMCIA SCSI adapters . . . 19
3.5.1 Diagnosing problems with SCSI adapters . . . 22
3.6 PCMCIA memory cards . . . 22
3.6.1 Using flash memory cards . . . 23
3.7 PCMCIA ATA/IDE card drives . . . 23
3.7.1 Diagnosing problems with ATA/IDE adapters . . . 24
3.8 Multifunction cards . . . 24
3.9 When is it safe to insert or eject a PCMCIA card? . . . 25
3.10 Card Services and Advanced Power Management . . . 25
3.11 How do I turn off a PCMCIA card without ejecting it? . . . 25
3.12 How do I unload the PCMCIA drivers? . . . 25

4 Advanced topics **26**
4.1 Resource allocation for PCMCIA devices . . . 26
4.2 How can I have separate device setups for home and work? . . . 26
4.3 Booting from a PCMCIA device . . . 27
4.3.1 The pcinitrd helper script . . . 28
4.3.2 Creating an initrd boot floppy . . . 28
4.3.3 Installing an initrd image on a non-Linux drive . . . 29

5 Dealing with unsupported cards **30**
5.1 Configuring unrecognized cards . . . 30
5.2 Adding support for an NE2000-compatible ethernet card . . . 31
5.3 PCMCIA floppy interface cards . . . 31
5.4 What's up with support for Xircom cards? . . . 32

6 Debugging tips and programming information **32**
6.1 How can I submit a helpful bug report? . . . 32
6.2 Low level PCMCIA debugging aids . . . 32

6.3 How do I write a Card Services driver for a new card? 33
6.4 Guidelines for PCMCIA client driver authors 33

1 General information and hardware requirements

1.1 Introduction

Card Services for Linux is a complete PCMCIA support package. It includes a set of loadable kernel modules that implement a version of the PCMCIA 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 PCMCIA cards, so cards can be inserted and ejected at any time.

This software is still 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 (`dhinds@hyper.stanford.edu`).

1.2 Copyright notice and disclaimer

Copyright (c) 1996, 1997 David A. Hinds

This document may be reproduced or distributed in any form without my prior permission. Modified versions of this document, including translations into other languages, may be freely distributed, provided that they are clearly identified as such, and this copyright is included intact.

This document may be included in commercial distributions without my prior consent. While it is not required, I would like to be informed of such usage. If you intend to incorporate this document in a published work, please contact me to make sure you have the latest available version.

This document is provided "as is", with no explicit or implied warranties. Use the information in this document at your own risk.

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

The current major release of Card Services is version 2.9, and minor updates or bug fixes are numbered 2.9.1, 2.9.2, and so on.

Source code for the latest version is available from `hyper.stanford.edu` in the `/pub/pcmcia` directory, as `pcmcia-cs-2.9.?.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 2.8.23. It is up to you to decide which version is more appropriate, but the `CHANGES` file will summarize the most important differences.

`hyper.stanford.edu` is mirrored at `sunsite.unc.edu` in `/pub/Linux/kernel/pcmcia`. I'll also try to upload major releases to `tsx-11.mit.edu` under `/pub/linux/packages/laptops/pcmcia` now and then.

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

1.4 What systems are supported?

This code should run on almost any Linux-capable laptop. All common PCMCIA controllers are supported, including Intel, Cirrus, Vadem, VLSI, Ricoh, and Databook chips. Custom controllers used in IBM and Toshiba laptops are also supported. PCMCIA card docks for desktop systems should work as long as they are the type that plugs directly into the ISA bus, rather than SCSI-to-PCMCIA or IDE-to-PCMCIA adapters.

The Motorola 6AHC05GA controller used in some Hyundai laptops is not supported. The custom PCMCIA controller in the HP Omnibook 600 is unsupported. PCI to CardBus bridge controllers (from SMC, Ricoh, Cirrus, and TI) are currently supported only in legacy 16-bit mode, and this support is still somewhat experimental.

1.5 What PCMCIA 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 new card be 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" PCMCIA code and other authors would contribute and maintain 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

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 PCMCIA programming and debugging. The Linux PCMCIA information page is at `http://hyper.stanford.edu/HyperNews/get/pcmcia/home.html`. 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 (and PCMCIA). There is also a searchable database of system configuration information.

2 Compilation, installation, and configuration

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 PCMCIA 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 PCMCIA package is not technically difficult, it does require some general Linux familiarity.

The following things should be installed on your system before you start installing PCMCIA:

- One of the following kernels: 1.2.8 through 1.2.13, 1.3.30, 1.3.37, 1.3.39 through 1.3.99, 1.99.* (i.e., pre-2.0), 2.0.*, or 2.1.*.
- A current set of module utilities.
- (Optional) the "Forms" X11 user interface toolkit.

The latest version requires a kernel version 1.2.8 or higher, or a development kernel 1.3.30 or higher. 1.3.38 is definitely broken, and 1.3.31 to 1.3.36 are untested. It also requires a relatively recent set of module utilities. There are no kernel patches specifically for PCMCIA.

You need to have a complete linux source tree for your kernel, not just an up-to-date kernel image, to compile the PCMCIA package. The PCMCIA 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 `sunsite.unc.edu` in `/pub/Linux/kernel/v2.0`, or from `tsx-11.mit.edu` in `/pub/linux/sources/system/v2.0`. Current module utilities can be found in the same place, in `modules-2.0.0.tgz`. Development kernels can be found in the corresponding `v2.1` subdirectories.

In the Linux source tree for 2.0 and 2.1 kernels, 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 2.1 kernel, be sure that you are using the right combination of shared libraries and module tools. The latest versions of the module utilities, as well as versions for older kernels, can be found at <`http://www.pi.se/blox/modules`>.

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.

If you want to use a PCMCIA token ring adapter, your kernel needs to 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 1.3.72 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, you should 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 you could change the serial device definition to:

```
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 Forms Library, which you will need to install before building `cardinfo`. A binary distribution is on `hyper.stanford.edu` in `/pub/pcmcia/extras`: there are both a.out and ELF versions of the library. You will also need to have all the normal X header files and libraries installed.

2.2 Installation

Here is a synopsis of the installation process:

- Unpack pcmcia-cs-2.9.?.tar.gz in /usr/src.
- Run "`make config`" in the new `pcmcia-cs-2.9.?` directory.
- Run "`make all`", then "`make install`".
- Customize the PCMCIA startup script and the option files in `/etc/pcmcia` for your site.

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.

If you are compiling the PCMCIA package for installation on another machine, specify an alternate target directory when prompted by the configure script. This should be an absolute path. All the PCMCIA tools 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.

If you are cross compiling on another machine, you may want to specify alternate names for the compiler and linker. This may also be helpful on mixed a.out and ELF systems. The script will also prompt for additional compiler flags for debugging.

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 configuration script will ask if you want the utilities compiled as safe or trusting: the default is to be safe.

There are a few kernel configuration options that affect the PCMCIA tools. The configuration script can deduce these from the running kernel (the 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.

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 extensions like `*.~1~`, `*.~2~`, and so on.

If you don't know what kind of PCMCIA controller chip you have, 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.

A user-level daemon processes card insertion and removal events. This is called `cardmgr`. It is similar in function to Barry Jaspan's `pcmciad` in earlier PCMCIA releases. `Cardmgr` reads a configuration file

describing known PCMCIA cards from `/etc/pcmcia/config`. This file also specifies what resources can be allocated for use by PCMCIA devices, and may need to be customized for your system. See the `pcmcia` man page for more information about this file.

2.3 Post-installation for systems using BSD init scripts

Some Linux distributions, including Slackware, use a BSD arrangement for system startup scripts. If `/etc/rc.d/rc.M` exists, your system is in this group. The script `rc.pcmcia`, installed in `/etc/rc.d`, controls starting up and shutting down the PCMCIA system. "`make install`" will use the `probe` command to determine your controller type and modify `rc.pcmcia` appropriately. You should add a line to your system startup file `/etc/rc.d/rc.M` to invoke the PCMCIA startup script, like:

```
/etc/rc.d/rc.pcmcia start
```

It does not really matter where you insert this line, as long as the PCMCIA drivers are started after `syslogd`.

2.4 Post-installation for systems using System V init scripts

Red Hat, Caldera, and Debian Linux have a System V-ish arrangement for system startup files. If you have a directory called `/etc/init.d` or `/etc/rc.d/init.d`, then your system is in this group. The `rc.pcmcia` script will be installed as `/etc/rc.d/init.d/pcmcia`, or `/etc/init.d/pcmcia`, as appropriate. There is no need to edit any of the startup scripts to enable PCMCIA: it will happen automatically.

If the `/etc/sysconfig` directory exists, then a separate configuration file, `/etc/sysconfig/pcmcia`, will be created for startup options. If you need to change any module options (like the `PCIC=` or `PCIC_OPTS=` settings), edit this config file rather than the actual PCMCIA startup script. This file will not be overwritten by subsequent installs.

Some previous releases used the `/etc/sysconfig/pcmcia-scripts` directory in place of `/etc/pcmcia` on these platforms. The current release instead uses `/etc/pcmcia` for all systems, and will move an existing `/etc/sysconfig/pcmcia-scripts` to `/etc/pcmcia`.

2.5 Site-specific configuration options

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.

- 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 HP Omnibook 4000C, exclude port 0x300-0x30f.
- 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 0x300-0x33f, irq 9, irq 10.
- 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 T4900 CT, exclude irq 5, port 0x2e0-0x2e8, port 0x330-0x338.
- 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.

Some PCMCIA controllers have optional features that may or may not be implemented in a particular system. It is generally impossible for a socket driver to detect if these features are implemented. Check the man page for your driver to see what optional features may be enabled.

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, as well as setting the `PCIC_OPTS` parameter to "`tcic_base=0x02c0`".

The low level socket drivers, `tcic` and `i82365`, have numerous bus timing parameters that may need to be adjusted for systems with particularly fast processors. Symptoms of timing problems include card recognition problems, lock-ups under heavy loads, high error rates, or poor device performance. Check the corresponding man pages for more details, but 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 and Databook TCIC-2 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 isn't a timing issue, but if you have more than one PCMCIA controller in your system or extra sockets in a docking station, the `i82365` module should be loaded with the `extra_sockets` parameter set to 1.

All these options should be configured by modifying the top of `/etc/rc.d/rc.pcmcia`. For example:

```
# Should be either i82365 or tcic
PCIC=i82365
# Put socket driver timing parameters here
PCIC_OPTS="cmd_time=12"
# Put pcmcia_core options here
CORE_OPTS="cis_speed=500"
```

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 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 PCMCIA controller. If this happens, edit `/etc/rc.d/rc.pcmcia` by hand as follows:

```
# Should be either i82365 or tcic
PCIC=i82365
# Put socket driver timing parameters here
PCIC_OPTS="wakeup=1"
```

2.6 Problems loading kernel modules

The configure script will normally ensure that the PCMCIA modules are compatible with your kernel. So, module loading problems generally indicate that the user has interfered with the normal installation process in some way. Some module loading errors are sent directly to the Linux console. Other errors are recorded in the system log file, normally `/usr/adm/messages` or `/var/log/messages`. Depending on your syslogd configuration, some messages may be written to other files, but they will usually still be under

/usr/adm or /var/log. To track down a problem, be sure to check both locations, to pin down which module is actually causing trouble.

Some of the PCMCIA 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 module.

If insmod reports "wrong version" errors, it means that the module was compiled for a different kernel version than your system is actually running. This might occur if modules compiled on one machine are copied to another machine with a different configuration, or if the kernel is reconfigured after PCMCIA is installed.

Another source of module loading errors is when the modules and kernel were compiled with different settings of CONFIG_MODVERSIONS. If a module with version checking is loaded against a kernel without version checking, insmod will complain about undefined symbols.

Finally, relatively recent binutils releases are incompatible with older versions of the module utilities, and can cause module version incompatibilities to be reported. The most common symptom is complaints about "gcc_compiled" being undefined. If you get these errors, upgrade to the latest module utilities, available from <http://www.pi.se/blox/modules>.

2.7 Problems with the card status change interrupt

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 PCMCIA 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.

2.8 Card identification problems

By default, the PCMCIA drivers allocate memory windows in the region 0xc0000-0xfffff, after probing this region for conflicts with ROM or other devices. This memory window is specified in

`/etc/pcmcia/config.opts`. The probe is done when the drivers first attempt to configure a new card. The probe procedure is not foolproof, so it is possible for a conflict to go unrecognized. If an address region that passes the probe is used by other devices in your system, cards may not be identified correctly. With chipsets that support it, conflicts can also result from BIOS shadowing in this region.

The classic symptom of a memory window configuration problem is that all cards will be misidentified as memory cards. In unusual cases, a memory window conflict can interfere with a crucial system service, resulting in lock-ups or reboots. If you suspect a memory window problem, first verify that ROM shadowing is disabled in your system's hardware setup. Finding a good window may require some experimentation. Some alternative windows to try are 0xd0000-0xdffff, 0xc9000-0xcffff, and 0xd8000-0xdffff.

If you have DOS 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 would be given as 0xd000). Be sure to add the extra digit back when making changes to `/etc/pcmcia/config.opts`.

If adjusting the memory window fails to solve a card identification problem, then it is likely to be due to a timing problem.

2.9 Why don't you distribute PCMCIA binaries?

For me, distributing binaries is a significant hassle. It is complicated because some features can only be selected at compile time, and because the PCMCIA 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 PCMCIA so that it can be used in the installation process for a particular Linux distribution. Each Linux distribution has its own procedures, 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.

2.10 Why is the PCMCIA package so darned big?

Well, first of all, it isn't actually that large. All the driver modules together take up about 200K of disk space. The utility programs add up to about 70K, and the stuff in /etc/pcmcia is about 30K. When running, the core PCMCIA modules take up 48K of system memory. The cardmgr daemon will generally be swapped out except when cards are inserted or removed. The total package size is not much different from DOS 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 PCMCIA 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 PCMCIA controller variants.

3 Usage and features

3.1 Tools for monitoring PCMCIA devices

The `cardmgr` daemon normally beeps when a card is inserted, and the tone of the beeps indicates the status of the newly inserted card. Two high beeps indicate the card was identified and configured successfully. A high beep followed by a lower beep indicates that the card was identified, but could not be configured for some reason. One low beep indicates that the card could not be identified.

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

```
Module:         #pages:  Used by:
ds                  2
i82365              3
pcmcia_core         7    [ds i82365]
```

All the PCMCIA modules and the `cardmgr` daemon send status messages to the system log. This will usually be `/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 contents of this file. If you are having trouble finding your system messages, check `/etc/syslogd.conf` to see how different classes of messages are handled.

`Cardmgr` also records some current device information for each socket in `/var/run/stab`. Here is a sample `/var/run/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).

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

```
Socket 0:
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
```

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`".

3.2 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/config`: 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 `/var/run/stab`. Each script constructs a "device address", that uniquely describes the device it has been asked to configure, in the `ADDRESS` 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` 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.

3.3 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 don't be surprised when you can't find them.

When a PCMCIA ethernet card is detected, it will be assigned the first free interface name, which will probably be `eth0`. `Cardmgr` will run the `/etc/pcmcia/network` script to configure the interface.

Do not configure your PCMCIA ethernet card in `/etc/rc.d/rc.inet1`, since the card may not be present when this script is executed. Comment out everything except the loopback stuff in `rc.inet1`. If your system has an automatic network configuration procedure, you should generally indicate that you do not have a network card installed. Instead, edit the `/etc/pcmcia/network.opts` file to match your local network setup. The `network` and `network.opts` scripts will be executed only when your ethernet card is actually present.

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
```

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 configured this way. 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.

3.3.1 Transceiver selection

The transceiver type can be selected in `network.opts` using the `IF_PORT` setting. This can either be a numeric value as in previous PCMCIA releases, 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)
```

Current releases of the 3c589 driver attempt to autodetect the network connection, but this doesn't seem to be completely functional yet. For autodetection to work, the network cable should be connected to the card when the card is configured. Alternatively, once the network is connected, you can force the driver to check the connection with:

```
ifconfig eth0 down up
```

3.3.2 Comments about specific cards

- With IBM CCAE and Socket EA cards, you need to pick the transceiver type (10base2, 10baseT, AUI) when the network device is configured. Make sure that the transceiver type reported in the system log matches your connection.
- The drivers for SMC, Megahertz, Ositech, and 3Com cards should autodetect the attached network type (10base2 or 10baseT). Setting the transceiver type when the driver is loaded serves to define the card's "first guess".
- 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. This driver also requires free IO ports in the range of 0xa20-0xa27. On some systems, the automatic IO port conflict checker will incorrectly determine that this port range is unavailable. In that case, the port check can be disabled by loading the `pcmcia_core` module with `probe_io=0`.

- Newer Linksys and D-Link 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. I am looking for beta testers for a Linux utility to perform this function.
- For WaveLAN wireless network adapters, Jean Tourrilhes (`jt@hplb.hpl.hp.com`) has put together a wireless HOWTO at `http://www-uk.hpl.hp.com/people/jt/Linux/Wavelan.html`.

3.3.3 Diagnosing problems with network adapters

- 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 section 2.5 for more information.
- If you get messages like "network unreachable" when you try to access the network, then you have probably set up `/etc/pcmcia/network.opts` incorrectly. On the other hand, mis-configured cards will usually fail silently.
- 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.

3.4 PCMCIA serial and modem devices

Linux serial devices are accessed via the `/dev/cua*` and `/dev/ttyS*` special device files. The `ttyS*` devices are for incoming connections, such as directly connected terminals. The `cua*` devices are for outgoing connections, such as modems. Each physical serial port has both a `ttyS` and a `cua` device file: it is up to

you to pick the appropriate device for your application. The configuration of a serial device can be examined and modified with the `setserial` command.

When a PCMCIA 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 `/var/run/stab`. The default serial device option script, `/etc/pcmcia/serial.opts`, will link the corresponding `cua*` device file to `/dev/modem` as a convenience.

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 PCMCIA 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 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 PCMCIA 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 `/var/run/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 "char/serial", "serial_cs"
```

3.4.1 Diagnosing problems with serial devices

- 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 section 5.1 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 PCMCIA 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`.

3.5 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`, `toaster_cs.c`, or `fdomain_cs.c`) with the normal Linux SCSI driver.

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`.

With 1.3.X and later kernels, the PCMCIA core drivers are able to find out from the kernel which SCSI devices are connected to a card. They will be listed in `/var/run/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.

With 1.2.X kernels, the PCMCIA drivers cannot automatically deduce which devices are associated with a particular SCSI adapter. Instead, if you have one normal SCSI device configuration, you may list these

devices in `/etc/pcmcia/scsi.opts`. For example, if you normally have a SCSI disk and a CD-ROM, you would use:

```
# For 1.2 kernels: list of attached devices
SCSI_DEVICES="sda scd0"
```

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. 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
```

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 `/var/run/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 only loaded when a PCMCIA SCSI adapter is detected. 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.

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'm not sure how hard it would be to write a driver; I don't think anyone has been able to obtain the technical information from Adaptec.

The (unsupported) Trantor SlimSCSI can be identified by the following:

```
Trantor / Adaptec APA-460 SlimSCSI
FCC ID: IE8T460
Shipped with SCSIworks! driver software
```

The (supported) Adaptec SlimSCSI can be identified by the following:

```
Adaptec APA-1460 SlimSCSI
FCC ID: FGT1460
P/N: 900100
Shipped with EZ-SCSI driver software
```

3.5.1 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"
```

- If you have compiled SCSI support as modules (`CONFIG_SCSI` is "m"), when configuring PCMCIA, you must explicitly specify that you want the SCSI drivers to be built. You must also modify `/etc/pcmcia/config` to load the SCSI modules before the appropriate `*_cs` driver is loaded.

3.6 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. 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
```

Some older memory cards, and most simple static RAM cards, lack a "Card Information Structure" (CIS), which is the scheme 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.

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`.

3.6.1 Using flash memory cards

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.

To use a flash memory card as an ordinary disk-like block device, first create a "flash translation layer" partition on the device with the `ftl_format` command:

```
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.

There are two major formats for flash memory cards: the "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.

3.7 PCMCIA ATA/IDE card drives

ATA/IDE drive support requires a 1.3.72 or higher kernel. The PCMCIA-specific part of the driver is `fixed_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 `fixed.opts` consist of either three or four fields: the current scheme, the socket number, the drive's serial number, and an optional partition number. As with SCSI devices, `fixed.opts` is first called for the entire device. If `fixed.opts` returns a list of partitions in the `PARTS` variable, the script will then be called for each partition.

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

```
case "$ADDRESS" in
*,*,*)
    PARTS="1"
    ;;
*,*,*,1)
    DO_FSTAB="y" ; DO_FSCK="y" ; DO_MOUNT="y"
```

```
    FSTYPE="msdos"
    OPTS=""
    MOUNTPT="/mnt"
    ;;
esac
```

Note that the default `fixed.opts` file has these lines but they are commented out. If you wish, you can have separate configurations for specific cards based on their serial numbers. To find out a drive's serial number, use the `ide_info` utility. Then, part of `fixed.opts` might look like:

```
case "$ADDRESS" in
*,*,Z4J60542)
    # This is my DOS stuff
    PARTS="1"
    ;;
*,*,Z4J60542,1)
    DO_FSTAB="y" ; DO_FSCK="y" ; DO_MOUNT="y"
    FSTYPE="msdos"
    OPTS=""
    MOUNTPT="/mnt"
    ;;
esac
```

3.7.1 Diagnosing problems with ATA/IDE adapters

- Some IDE drives violate the PCMCIA specification by requiring a longer time to spin up than the maximum allowed card setup time. To use these cards, load the `pcmcia_core` module with:

  ```
  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.

3.8 Multifunction cards

Starting with the 1.3.73 Linux kernel, a single interrupt can be shared by several drivers, such as the serial driver and an ethernet driver. When using a multifunction card under a newer kernel, all card functions can be used without loading and unloading drivers.

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, Linksys) properly support simultaneous access, but others (Megahertz in particular) do not.

Earlier kernels do not support interrupt sharing between different device drivers, so it is not possible for the PCMCIA drivers to configure this card for simultaneous ethernet and modem access. The ethernet and

serial drivers are both loaded automatically. However, the ethernet driver "owns" the card interrupt by default. To use the modem, you can unload the ethernet driver and reconfigure the serial port by doing something like:

```
ifconfig eth0 down
rmmod 3c589_cs
setserial /dev/modem autoconfig auto_irq
setserial /dev/modem
```

The second `setserial` should verify that the port has been configured to use the interrupt previously used by the ethernet driver.

3.9 When is it safe to insert or eject a PCMCIA card?

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.

3.10 Card Services and Advanced Power Management

Card Services can be compiled with support for APM (Advanced Power Management) if you've installed this package on your system. APM is incorporated into 1.3.46 and later kernels. It is currently being maintained by Rick Faith (faith@cs.unc.edu), and APM tools can be obtained from `ftp.cs.unc.edu` in `/pub/users/faith/linux`. The PCMCIA modules will automatically be configured for APM if a compatible version is detected on your system.

Without resorting to APM, you can do "`cardctl suspend`" before suspending your laptop, and "`cardctl resume`" after resuming, to properly shut down and restart your PCMCIA cards. This will not work with a PCMCIA 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, use "cardctl eject" prior to suspending the system.

3.11 How do I turn off a PCMCIA card without ejecting it?

Use either the `cardctl` or `cardinfo` command. "`cardctl suspend #`" will suspend one socket, and turn off its power. The corresponding `resume` command will wake up the card in its previous state.

3.12 How do I unload the PCMCIA drivers?

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 PCMCIA 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 Advanced topics

4.1 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 PCMCIA card to use. The `/etc/pcmcia/config.opts` file allows you to specify ranges of ports available for use by all PCMCIA devices, or to exclude ranges that conflict with other devices.

After modifying `/etc/pcmcia/config.opts`, you can restart `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.

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

This is fairly easy using PCMCIA "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 PCMCIA 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 choose 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 PCMCIA scheme setting, run:

```
cardctl scheme
```

4.3 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.

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.

4.3.1 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 PCMCIA 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 PCMCIA configuration files will be installed under `/etc` in the image, and can also be customized. See the man page for `pcinitrd` for more information.

4.3.2 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
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.

4.3.3 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.

5 Dealing with unsupported cards

5.1 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.

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]: version info: "MEGAHERTZ", "XJ2288", "V.34 PCMCIA MODEM"
```

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"
```

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.

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.

5.2 Adding support for an NE2000-compatible ethernet card

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 section 5.1 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 `modules/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` 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.

5.3 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.

5.4 What's up with support for Xircom cards?

A driver for Xircom ethernet and ethernet/modem cards is included in the current PCMCIA package, thanks to the work of Werner Koch. I've set up a HyperNews forum specifically for discussion of Xircom driver development, at `http://hyper.stanford.edu/HyperNews/get/pcmcia/xircom.html`.

For a long time, Xircom cards were not supported because Xircom had a company policy of not disclosing technical information about their cards. However, they have relaxed their rules, and now, they do distribute driver information.

6 Debugging tips and programming information

6.1 How can I submit a helpful bug report?

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 type, and the output of the `probe` command.
- What PCMCIA cards you are using.
- Your Linux kernel version, and PCMCIA driver version.
- Any changes you've made to the startup files in `/etc/pcmcia`, or to the PCMCIA startup script.
- All PCMCIA-related messages in your system log file.

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 your problem involves a kernel fault, the register dump from the fault is only useful if you can track down the fault address, EIP. If it is in the main kernel, look up the address in `System.map` to identify the function at fault. If the fault is in a loadable module, it is a bit harder to trace. With the current module tools, "`ksyms -m`" will report the base address of each loadable module. Pick the module that contains the EIP address, and subtract its base address from EIP to get an offset inside that module. Then, run `gdb` on that module, and look up the offset with the `list` command. This will only work if you've compiled that module with `-g` to include debugging information.

If you do not have web access, bug reports can be sent to me at `dhinds@hyper.stanford.edu`. However, I prefer that bug reports be posted to my web site, so that they can be seen by others.

6.2 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.

There are a few debugging tools in the `debug_tools/` subdirectory of the PCMCIA distribution. The `dump_tcic` and `dump_i365` utilities generate complete register dumps of the PCMCIA controllers, and decode a lot of the register information. They are most useful if you have access to a datasheet for the corresponding controller chip. The `dump_tuples` utility lists a card's CIS (Card Information Structure), and decodes some 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. It can be bound to any PCMCIA card, and does not interfere with other drivers. It can be used to directly access any card's attribute memory or common memory.

6.3 How do I write a Card Services driver for a new card?

The Linux PCMCIA Programmer's Guide is the best documentation for the Linux PCMCIA interface. The latest version is always available from `hyper.stanford.edu` in `/pub/pcmcia/doc`, or on the web at `http://hyper.stanford.edu/HyperNews/get/pcmcia/home.html`.

For devices that are close relatives of normal ISA devices, you'll 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.

I've written a skeleton driver with lots of comments that explains a lot of how a driver communicates with Card Services; you'll find this in the PCMCIA source distribution in `modules/skeleton.c`.

6.4 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 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
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.

I will accept client drivers prepared according to this specification and place them in the `/pub/pcmcia/contrib` directory on my FTP server, `hyper.stanford.edu`. The README in this directory will describe how to unpack a contributed driver.

The PCMCIA 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 PCMCIA package. I will try to notify authors of contributed drivers of changes that require updates to their drivers.