Modem−HOWTO

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Modem−HOWTO

David S.Lawyer `mailto:dave@lafn.org`

v0.16, March 2001

Help with selecting, connecting, configuring, trouble−shooting, and understanding modems for a PC. See Serial−HOWTO for multiport serial boards.

---

5. Configuring Overview

6. Configuring the Serial Port

7. Modem Configuration (excluding serial port)

8. Serial Port Devices /dev/ttyS2, etc.

9. Interesting Programs You Should Know About

10. Trying Out Your Modem (Dialing Out)

11. Dial−In

12. Uugetty for Dial−In (from the old Serial−HOWTO)

13. What Speed Should I Use with My Modem?

14. Communications Programs And Utilities

15. Troubleshooting

22. Appendix E: Other Types of Modems

23. Appendix F: Fax pixels (dots)

24. Appendix G: Antique Modems

---

1. Introduction

1.1 DSL, Cable, and ISDN Modems in other HOWTOs

This HOWTO covers conventional analog modems for PCs on either the ISA or PCI bus. For other types of modems:

- DSL modems: see the mini−howto: ADSL
- Cable−Modems−HOWTO (was once a LDP mini−Howto) http://www.cs.unm.edu/~vuksan/linux/Cable−Modem.html
- Cable−Modem−Providers−HOWTO
- ISDN Howto (not a LDP Howto) http://sdb.suse.de/sdb/en/html/isdn.html: drivers for ISDN "Modems". Much related info on this is in German. For a tutorial on ISDN see http://public.swbell.net/ISDN/overview.html

See also Appendix D: Other Types of Modems

1.2 Also not covered: PCMCIA Modems, PPP

For modems on the PCMCIA bus see the PCMCIA−HOWTO: PCMCIA serial and modem devices. This HOWTO also doesn't cover PPP (used to connect to the Internet via a modem) or communication programs. Except it does show how to use communication programs to test that your modem works OK and can make phone calls. If you want to use a modem to connect to the Internet then you need to set up PPP. There's a lot of documentation for PPP (including a PPP−HOWTO). More documentation should be found in /usr/doc/ppp, /usr/share/doc/ppp or the like.

1.3 Copyright, Disclaimer, Trademarks, & Credits

Copyright

Disclaimer

While I haven't intentionally tried to mislead you, there are likely a number of errors in this document. Please let me know about them. Since this is free documentation, it should be obvious that I cannot be held legally responsible for any errors.

Trademarks.

Any brand names (starts with a capital letter) should be assumed to be a trademark). Such trademarks belong to their respective owners.

"Hayes" is a trademark of Microcomputer Products Inc. I use "winmodem" to mean any modem which requires MS−Windows and not in the trademark sense. All other trademarks belong to their respective owners.

Credits

The following is only a rough approximation of how this this document (as of 2000) was created: About 1/4 of the material here was lifted directly from Serial−HOWTO v. 1.11 (1997) by Greg Hankins. mailto:gregh@twoguys.org (with his permission). About another 1/4 was taken from that Serial−HOWTO and revised. The remaining 1/2 is newly created by the new author: David S. Lawyer `mailto:dave@lafn.org`.

1.4 Contacting the Author

Since I don't follow the many different brands/models of modems please don't email me with questions about them (or suggestions of which one to buy). If you are interested in a certain model (to find out if it works under Linux, etc.) see the huge list at Web Sites. Also, please don't ask me how to configure a modem unless you've looked over this HOWTO and still can't do it. I've no personal experience with software−based modems.

Please let me know of any errors in facts, opinions, logic, spelling, grammar, clarity, links, etc. But first, if the date is over a month old, check to see that you have the latest version. Please send me any other info that you think belongs in this document.

1.5 New Versions of this HOWTO

New versions of this Modem−HOWTO come out every month or two since modem situation is rapidly changing (and since I'm still learning). Your problem might be solved in the latest version. It will be available to browse and/or download at LDP mirror sites. For a list of such sites see: http://www.linuxdoc.org/mirrors.html If you only want to quickly compare the date of this the version v0.16, March 2001 with the date of the latest version go to: http://www.linuxdoc.org/HOWTO/Modem−HOWTO.html

1.6 New in Recent Versions

v0.16 March 2001 New url for winmodem.html, more obsolete protocols v0.15 March 2001 Revision of Dial−In section (incl. problems with agetty, interoperability with MS, VNC), blacklist, AT−cmds on Internet, broken links fixed

1.7 What is a Modem ?

A modem is a device that lets one send digital signals over an ordinary telephone line not designed for digital signals. If telephone lines were all digital then you wouldn't need a modem. It permits your computer to connect to and communicate with the rest of the world. When you use a modem, you normally use a communication program or web browser to utilize the modem and dial−out on a telephone line. Advanced modem users can set things up so that others may phone in to them and use their computer. This is called "dial−in".

There are three basic types of modems for a PC: external, internal, and built−in. The external sets on your desk outside the PC while the other two types are not visible since they're inside the PC. The external modem plugs into a connector on the back of the PC known as a "serial port". The internal modem is a card that is inserted inside the computer. The built−in modem is part of the motherboard and is thus built into the computer. It's is just like an internal modem except it can't be removed or replaced. As of 2001, built−in modems are primarily for laptops. What is said in this HOWTO regarding internal modems will generally apply also to built−in modems.

For a more detailed comparison see External vs. Internal. When you get an internal (or built−in) modem, you also get a dedicated serial port (which can only be used with the modem and not with anything else such as another modem or a printer). In Linux, the serial ports are named ttyS0, ttyS1, etc. (usually corresponding respectively to COM1, COM2, etc. in Dos/Windows).

The serial port is not to be confused with the "Universal Serial Bus" (USB) which uses a special modular connector and may be used to connect an external modem. See Modem & Serial Port Basics for more details on modems and serial ports.

Modems usually include the ability to send Faxes (Fax Modems). See Fax for a list of fax software. "Voice" modems can work like an automatic answering machine and handle voicemail. See Voicemail.

1.8 Quick Install

External Modem Install

With a straight−thru or modem cable, connect the modem to an unused serial port on the PC. Make sure you know the name of the serial port: in most cases COM1 is ttyS0, COM2 is ttyS1, etc. You may need to check the BIOS setup menu to determine this. Plug in the power cord to provide power to the modem. See All Modems for further instructions.

Internal Modems (Both ISA and PCI)

The first thing to do is to make sure that the modem will work under Linux since (as of late 2000) most newer modems don't. See modem list. If the modem is both PnP and directly supported by the serial driver (kernel 2.4 +) then there is no configuring for you to do since the driver will configure it.

To physically install a modem card, remove the cover of the PC by removing some screws (perhaps screw size 6−32 in the U.S.). Find a matching vacant slot for it next to the other adapter cards. Before inserting the card in the slot, remove a strip on the back of the PC so that the telephone jacks on the card will be accessible from the back of the PC. Then carefully align the card with the slot and push the card all the way down into the slot. Attach the card with a mounting screw (usually 3mm, .5mm pitch −−don't use the wrong size).

If you have a modem that is not a winmodem (see Software−based Modems (winmodems)) and you must configure it yourself (the driver doesn't do it) then you first need to decide which ttySx to assign it to. Pick a ttySx that is not already in use by other serial ports. Then you have the problem of setting an IRQ number and IO address. For PnP modems: If the BIOS has already set these in the physical device (which a PnP BIOS will do if it thinks you don't have a PnP OS) then you need to determine the IRQ and IO address and then tell this to "setserial". In other cases you may have some choice of IRQs and IO addresses (including the case where you are able to change what the BIOS has set). See Choosing Serial IRQs and Choosing Addresses. For ISA modems there are standard IO addresses to use (corresponding to the ttySx). PCI modems seem to use different IO addresses so as not to conflict with ISA modems. For example you may find it feasible to use /dev/ttyS2 at IO address 0x3e8 and IRQ 11.

ISA Modems: What IOs and IRQs may be used?

For old modems with jumpers look at the manual (or jumpers if they say). If the BIOS has already configured the ISA modem then "isapnp −−dumpregs" should show it. If you need to set or change them use "isapnp". Use the "pnpdump" to see what you changes are possible.

PCI Modems: What IOs and IRQs have been set?

For PCI, the BIOS almost always sets the IRQ any may have set the IO address as well. To see how it's set use "lspci −v" or look in /proc/bus/pci. If more than one IO address is shown, the first one is more likely to be it. You can't change the IRQ (at least not with "setpci") If you must, change the IO address with "setpci" by changing the BASE_ADDRESS_0 or the like. The _0 (or _1) after BASE_ADDRESS must be the correct register.

Both PCI and ISA: Use setserial to tell the driver

You must find the file where "setserial" is run at boot−time and add a line something like: "setserial /dev/ttyS2 irq 5 port 0x0b8". For setserial v2.15 and later the results of running "setserial" on the command line may (or may not) be saved to /etc/serial.conf so that it runs each time you boot. See What is Setserial for more info. See the next subsection All Modems for further instructions on quick installation.

Use MS Windows to set the BIOS (A last resort method)

If you are using the BIOS to configure you may use MS Windows9x to "force" the BIOS to set a certain IRQ and/or IO. It can set them into the PnP BIOS's flash memory where they will be used to configure for Linux as well as Windows. See "Plug−and−Play−HOWTO and search for "forced" (occurs in several places). For Windows3.x you can do the same thing using the ICU under Windows 3.x. A few modems have a way to disable PnP in the modem hardware using software (under Windows) that came with the modem.

All Modems

Plug the modem into a telephone line. Then configure a communication program such as minicom or a ppp program (such as wvdial). Set the serial port speed to a baud rate a few times higher than the bit rate of your modem. See Speed Table for the "best" speeds to use. Tell it the full name of your serial port such as /dev/ttyS1. Set hardware flow control (RTS/CTS).

Minicom is the easiest to set up and to use to test your modem. But if you are lucky you may get ppp to work the first time and not need to bother with minicom. With minicom you may simply type the command: AT and hit enter (before dialing) to check that your modem is there and responds with OK.

---

2. Modems for a Linux PC

2.1 External vs. Internal

A modem for a PC may be either internal or external. The internal one is installed inside of your PC (you must remove screws, etc. to install it) and the external one just plugs into a serial port connector on a PC. Internal modems are less expensive, are less likely to to suffer data loss due to buffer overrun, usually use less electricity, and use up no space on your desk.

External modems are usually easier to install and usually require less configuration. They have lights which may give you a clue as to what is happening and aid in troubleshooting. The fact that the serial port and modem can be physically separated also aids in troubleshooting. External modems are easy to move to another computer.

Unfortunately most external modems have no switch to turn off the power supply when not in use and thus are likely to consume a little electricity even when turned off (unless you unplug the power supply from the wall). Each watt they draw costs you about $1/yr. Another possible disadvantage of an external is that you will be forced to use an existing serial port which may not support a speed of over 115,200 bps (although as of late 1998 and late 2000 most new internal modems don't either −−but some do). If a new internal modem had say a 16650 UART it would put less load on the CPU.

Internal modems present a special problem for Linux, but will work just as well as external modems provided you avoid the high percentage of them that will work only for MS Windows, and also provided that you spend time (sometimes a lot of time) to configure them correctly. Some of the modems which will work only under MS Windows are, unfortunately, not labeled as such. If you buy a new one, make sure that you can return it for a refund if it will not work under Linux.

While most new modems are plug−and−play you have various ways to deal with the PnP configuring:

- The serial driver does it all for you (more likely for a PCI modem)
- Use the "isapnp" program
- Let a PnP BIOS do the configuring

Each of the above has shortcomings. Isapnp documentation is difficult to understand although reading the Plug−and−Play−HOWTO (long) will aid in understanding it. If you want the PnP BIOS to do the configuring, all you need to do is to make sure that it knows you don't have a PnP operating system. But it may not do it correctly and you may need to find out what it's done see What is set in my serial port hardware?.

There are many Linux users that say that it's a lot simpler just to get an external modem and plug it in. But since new peripherals are PnP today, you will sometime need to deal with it, so why delay the inevitable? Still, the most expedient (and expensive) solution is an external modem (if you have a free serial port).

2.2 External Modems

PnP External Modems

Many external modems are labeled "Plug and Play" (PnP) but they should all work fine as non−PnP modems. While the serial port itself may need to be configured (IRQ number and IO address) unless the default configuration is OK an external modem uses no such IRQ/IO configuration. You just plug the modem into the serial port. Since you usually plug the modem into a serial port (and connect it to power).

How can an external modem be called PnP since it can't be configured by PnP? Well, it has a special PnP identification built into it that can be read (thru the serial port) by a PnP operating system. Such an operating system would then know that you have a modem on a certain port and would also know the model number. Then you might not need to configure application programs by telling them what port the modem is on (such as /dev/ttyS2 or COM3). But since you don't have such a PnP operating system you will need to configure your application program manually by giving it the /dev id (such as /dev/ttyS2).

Cabling & Installation

Connecting an external modem is simple compared to connecting most other devices to a serial port that require various types of "null modem" cables (which will not work for modems). Modems use straight through cable, with no pins crossed over. Most computer stores should have this. Make sure you get the correct gender and number of pins. Hook up your modem to one of your serial ports. If you are willing to accept the default IRQ and IO address of the port you connect it to, then you are ready to start your communication program and configure the modem itself.

What the Lights (LED's) Mean (for some modems)

- TM Test Modem

- AA Auto Answer (If on, your modem will answer an incoming call)
- RD Receive Data line = RxD
- SD Send Data line = TxD
- TR data Terminal Ready = DTR (set by your PC)
- RI Ring Indicator (If on, someone is "ringing" your modem)
- OH Off Hook (If off, your modem has hung up the phone line)
- MR Modem Ready = DSR ??
- EC Error Correction
- DC Data Compression
- HS High Speed (for this modem)

2.3 Internal Modems

An internal modem is installed in a PC by taking off the cover of the PC and inserting the modem card into a vacant slot on the motherboard. There are modems for the ISA slots and others for the PCI slots. Some new PC don't have any ISA slots. While external modems plug into the serial port (via a short cable) the internal modems have the serial port built into the modem. In other words, the modem card is both a serial port and a modem.

Setting the IO address and IRQ for a serial port was formerly done by jumpers on the card. These are little black rectangular "cubes" about 5x4x2 mm in size which push in over pins on the card. Plug−and−Play modems (actually the serial port part of the modems) don't use jumpers for setting these but instead are configured by sending configuration commands to them over the bus inside the computer. Such configuration commands can be sent by a PnP BIOS, by the isapnp program (for the ISA bus only), or by newer serial device drivers for certain modems. Under Linux you have a choice of how to configure the ones that don't get io−irq configured by the serial driver.

1. Use "isapnp" which may be run automatically at every boot−time
2. Let a PnP BIOS do it, and then maybe tell setserial the IO and IRQ

2.4 Software−based Modems (winmodems)

Introduction software modems (winmodems)

Software modems turn over some (or even almost all) of the work of the modem to the main processor (CPU) chip of your computer (such as a Pentium chip). This requires special software (a modem driver) to do the job. Until late 1999, such software was released only for MS Windows and wouldn't work with Linux. Even worse was that the maker of the modem kept the interface to the modem secret so that no one could write a Linux driver for it (even though a few volunteers were willing to write Linux drivers). With few exceptions, this is still true today (late 2000). Also, there is no standard interface so that different brands/models of software−modems need different drivers (unless the different brands/models happen to use the same chipset internally).

A third name for a software modem (used by MS) is "driver−based modem". The conventional hardware−based modem (that works with Linux) doesn't need a modem driver (but does use the Linux serial driver) After about mid−1998 most new internal modems were winmodems and would work only for MS Windows.

Linmodems

Finally in late 1999 two software−based modems appeared that could work under Linux and were sometimes called "linmodems". Lucent Technologies (LT) unofficially released a Linux binary−only code to support its PCI modems. PC−TEL (includes "Zoltrix") introduced a new software−based modem for Linux. There is a GPL'ed driver being developed for the Modem Silicon Operation MD563X HaM chipset (nee Ambient division of Cirrus Logic). Will other companies follow these leads and thus create "linmodems"? For a list of modems which work/don't_work under Linux see modem list. Links to "linmodem" drivers may also be found there. A project to get winmodems to work under Linux is at http://linmodems.org. They also have a mailing list.

There is some effort underway at reverse−engineering with at least one report of a winmodem that has been made to work under Linux (but not yet with full functionality). So by the time you read this there may be more linmodems.

For details of how to get some winmodems to work under Linux see the Linmodem−HOWTO (and/or Winmodems−and−Linux−HOWTO which is not as well written). If code is made available to operate a "winmodem" under Linux, then one may call it a "linmodem". Is it still a "winmodem"? Well, it's still a software−based modem. The term "Winmodem" is also a trademark for a certain model of "winmodem".

Software−based modem types

There are two basic types of software modems. In one type the software does almost all of the work. The other is where the software only does the "control" operations (which is everything except processing the digital waveshapes −−to be explained later). Since the hardware doesn't do the control it's called a "controllerless" modem. The first type is a all−software modem (sometimes just called a software modem).

For both of these types there must be analog hardware in the modem to generate an electrical waveshape to send out the phone line. It's generated from a digital signal (which is sort of a "digital waveshape"). It's something like the digital electronics creates a lot of discrete points on graph paper and then the modem draws a smooth curve thru them. There must also be hardware to convert the incoming waveshape to digital. Then this digital waveshape must be converted to a data byte stream. The modem can't just send this data byte stream to the PC but must first do decompression, error correction, and convert from serial to the parallel bus of the computer.

The difference between the two types of software−based modems is where these digital waveshapes are processed (generated and interpreted). In the all−software modem this waveshape processing is done in the CPU using a Host Signal Processor (HSP). In the controllerless modem it's done in the modem but all other digital work is done by the CPU (data compression, AT−commands, etc.) For example the Rockwell HCF (Host Controlled Family) does this. If the software that does these tasks could be ported to Linux and then there wouldn't be a major problem.

Is this modem a software modem?

How do you determine if an internal modem will work under Linux? First see if the name, description of it, or even the name of the MS Windows driver for it indicates it's a software modem: HSP, HCF, HSF, controllerless, host−controlled, host−based, and soft−... modem. If it's one of these modem it will only work for the few cases (so far) where a Linux driver is available.

If you don't know the model of the modem and you also have Windows on your Linux PC, click on the "Modem" icon in the "Control Panel". Then check out the modem list (see Web Sites. If the above doesn't work (or isn't feasible), you can look at the package it came in (or a manual) find the section on the package that says something like "Minimum System Requirements" or just "System Requirements". It may be in fine print. Read it closely. If Windows or a Pentium CPU is listed as one of the requirements then it is not too likely to work under Linux.

Otherwise, it may work under Linux if it fails to state explicitly that you must have Windows. By saying it's "designed for Windows" it may only mean that it fully supports Microsoft's plug−and−play which is OK since Linux uses the same plug−and−play specs (but it's harder to configure under Linux). Being "designed for Windows" thus gives no clue as to whether or not it will work under Linux. You might check the Website of the manufacturer or inquire via email. I once saw a web−page that specifically stated that one model worked under Linux while implying that another model didn't.

Should I get a software modem?

Only if you know there is a Linux driver for it that works OK. Besides the problems of getting a driver, what are the pros and cons of software modems? Since the software modem uses the CPU to do much of its work, the software modem requires less on−board electronics and thus costs less. At the same time, the CPU is heavily loaded by the modem which may result in slower operation. This is especially true if other CPU−intensive tasks are running at the same time the modem is being used. Of course when you're not using the software modem there is no degradation in performance at all.

Is the cost savings worth it? In some cases yes, especially if you seldom use the modem and/or are not running any other CPU intensive tasks when the modem is in use. Thus there are cases where use of a software modem is economically justified. The savings in modem cost could be used for a better CPU which would speed things up a little. But the on−board electronics of a modem can do the job more efficiently than a general purpose CPU (except that it's not efficient when it's not in use). So if you use the modem a lot it's probably better to avoid software modems (and then you can use a less powerful CPU :−).

2.5 PCI Modems

A PCI modem card is one which inserts into a PCI−bus slot on the motherboard of a PC. While most PCI winmodems will not work under Linux (no driver available) other PCI modems mostly work under Linux. The Linux serial driver is being modified to support certain PCI modem cards (but not winmodems). If the Linux serial driver supports it then the driver will set up the PnP configuration for you. See PCI Bus Support Underway If no special support is in the Linux serial driver but it may still work OK but you have to do some work to configure it.

2.6 Which Internal Modems might not work with Linux

- Software−based Modems (winmodems) only work in rare cases where a Linux driver is available.
- MWave and DSP Modems might work, but only if you first start Windows/Dos each time you power on your PC
- Modems with RPI (Rockwell) drivers work but with reduced performance

MWave and DSP Modems

Such modems use DSP's (Digital Signal Processors) which are programmed by driver which must be downloaded from the hard disk to the DSP's memory just before using the modem. Unfortunately, such downloading is normally done by Dos/Windows programs (which doesn't work for Linux). But there has been substantial success in getting some of these modems to work with Linux. For example, there is a Linux driver available to run a Lucent (DSP) modem.

Ordinary modems that work fine with Linux (without needing a driver for the modem) often have a DSP too (and may mention this on the packaging), but the program that runs the DSP is stored inside the modem. This is not a "DSP modem" in the sense of this section. An example of a DSP modem is IBM's Aptiva MWAVE.

If a DSP modem modem simulates a serial port, then it may be usable with Linux provided you're willing/able to boot from DOS. You must have Dos/Windows on the same PC. You first install the driver under DOS (using DOS and not Window drivers). Then start Dos/Windows and start the driver for the modem so as to program the DSP. Then without turning off the computer, go into Linux.

One may write a "batch" file (actually a script) to do this. Here is an example but you must modify it to suit your situation.

```
rem mwave is a batch file supplied by the modem maker
call c:\mww\dll\mwave start
rem loadlin.exe is a DOS program that will boot Linux from DOS (See
rem Config−HOWTO).
c:\linux\loadlin f:\vmlinuz root=/dev/hda3 ro
```

One may create an icon for the Window's desktop which points to such a batch file and set the icon properties to "Run in MSDOS Mode". Then by clicking on this icon one sets up the modem and goes to Linux. Another possible way to boot Linux from DOS is to press CTRL−ALT−DEL and tell it to reboot (assuming that you have set things up so that you can boot directly into Linux). The modem remains on the same com port (same IO address) that it used under DOS.

The Newcom ifx modem needs a small kernel patch to work correctly since its simulation of a serial port is non−standard. The patch and other info for using this modem with Linux is at http://maalox.pharmacy.ohio−state.edu/~ejolson/linux/newcom.html.

Rockwell (RPI) Drivers

Some older Rockwell chips need Rockwell RPI (Rockwell Protocol Interface) drivers. They can still be used with Linux even though the driver software works only under MS Windows. This is because the MS Windows software which you don't have does only compression and error correction. If you are willing to operate the modem without compression and error correction then it's feasible to use it with Linux. To do this you will need to disable RPI by sending the modem (via the initialization string) a "RPI disable" command each time you power on your modem. On my modem this command was +H0. Not having data compression available makes it slower to get webpages but is just as fast when downloading files that are already compressed.

3. Modem Pools, Digital Modems

A modem pool is a number of modems on the same card (such as a multiport modem card) or many modems in an external chassis (something like an external modem). The modems may be analog modems similar to modems used for home/office PCs (can't send at 56k even if they are "56k modems"). They also could be "digital modems" which can send at nearly 56k (if you have a good line). The "digital modems" require a digital connection to the telephone line and don't use any serial ports at all. All of these modem pools will require that you install special drivers for them.

3.1 Analog Modem Pools, Multiport Modem Cards

These are just many analog modems (the common home/office modem) provided either on a plug−in card or in an external chassis. Each modem comes with a built−in serial port. There is usually a system of sharing interrupts or of handling interrupts by their own electronics, thus removing much of this burden from the CPU. Note that these modems are not "digital modems" and will thus not be able to use 56k for people who dial−in.

Here is a list of some companies that make multiport modem cards. 8 modems/card is common. The cards listed claim to work with Linux and the websites should point you to a driver for them.

Multiport Modem Cards:

- MultiModemISI by Multi−Tech Systems. 56k or 33.6k, PCI or ISA, 4 or 8 ports. ISDN/56k hybrids. http://www.multitech.com/products/
- RAStel by Moreton Bay Products. 56k PCI or ISA, 4 or 8 ports. Also 2 modems + 2 vacant serial ports.
  http://www.moretonbay.com.au/MBWEB/product/rastel/rastel.htm
- RocketModem by Comtrol. ISA 33.6k, 4 or 8 port.
  http://www.comtrol.com/SALES/SPECS/Rmodem.htm
- AccelePort (RAS Family) by Digi. http:/www.dgii.com/digi.cfm?p=940564.pi.prd.00000046

3.2 Digital Modems

"digital modems" are much different than the analog modems that most people use in their PCs. They require a digital connection to the telephone line and don't use serial ports for the interface to the computer. Instead, they interface directly to the PC bus via a special card (which may also contain the "digital modems"). They are able to send at near 56k, something no analog modem can do. They are often a component of "remote access servers" (RASs) or "digital modem pools"

The cables from the phone company that carry digital signals have been designed for high bandwidth so that the same cable carries multiple telephone calls. It's done by "time−division multiplexing". So the first task to be done is to separate the phone calls and send each phone call to its own "digital modem". There is also the task in the reverse direction of combining all of the calls onto a single line. These tasks are done by what is sometimes called a "... concentrator".

The digital modem gets the digital signal from the telephone company. It converts the waveshape it represents back to the same data bytes that were sent from the sending PC. It puts these bytes on its bus (likely sending it to a buffer in memory). Likewise, it handles sending digital signals in the opposite direction to a digital telephone line. Thus it only makes digital−to−digital conversions and doesn't deal in analog at all.

It thus is not really a modem at all since it doesn't modulate any analog carrier. So the name "digital modem" is a misnomer but it does do the job formerly done by modems. Thus some "digital modems" call themselves "digital signal processors", or "remote access servers", etc. and may not even mention the word "modem". This is technically correct terminology.

Such a system may be a stand−alone proprietary server, a chassis containing digital modems that connects to a PC via a special interface card, or just a card itself. Digi calls one such card a "remote access server concentrator adapter". One incomplete description of what is needed to become an ISP is: See What do I need to be an ISP?. Cyclades promotes their own products here so please do comparison shopping before buying anything.

---

4. Serial Port and Modem Basics

You don't have to understand the basics to use and install a modem. But understanding it may help to determine what is wrong if you run into problems. After reading this section, if you want to understand it even better you may want to see How Modems Work in this document (not yet complete). More details on the serial port (including much of this section) will be found in Serial−HOWTO.

4.1 Modem Converts Digital to Analog (and conversely)

Most all telephone main lines are digital already but the lines leading to your house (or business) are usually analog which means that they were designed to transmit a voltage wave which is an exact replica of the sound wave coming out of your mouth. Such a voltage wave is called "analog". If viewed on an oscilloscope it looks like a sine wave of varying frequency and amplitude. A digital signal is like a square wave. For example 3 v (volts) might be a 1−bit and 0 v could be a 0−bit. For most serial ports (used by external modems) +12 v is a 0−bit and −12 v is a 1−bit (some are + or − 5 v).

To send data from your computer over the phone line, the modem takes the digital signal from your computer and converts it to "analog". It does this by both creating an analog sine wave and then "MODulating" it. Since the result still represents digital data, it could also be called a digital signal instead of analog. But it looks something like an analog signal and almost everyone calls it analog. At the other end of the phone line another modem "DEModulates" this signal and the pure digital signal is recovered. Put together the "mod" and "dem" parts of the two words above and you get "modem" (if you drop one of the two d's). A "modem" is thus a MODulator−DEModulator. Just what modulation is may be found in the section Modulation Details.

4.2 What is a Serial Port ?

Intro to Serial

The serial port is an I/O (Input/Output) device. Since modems have a serial port between them and the computer, it's necessary to understand the serial port as well as the modem.

Most PC's have one or two serial ports. Each has a 9−pin connector (sometimes 25−pin) on the back of the computer. Computer programs can send data (bytes) to the transmit pin (output) and receive bytes from the receive pin (input). The other pins are for control purposes and ground.

The serial port is much more than just a connector. It converts the data from parallel to serial and changes the electrical representation of the data. Inside the computer, data bits flow in parallel (using many wires at the same time). Serial flow is a stream of bits over a single wire (such as on the transmit or receive pin of the serial connector). For the serial port to create such a flow, it must convert data from parallel (inside the computer) to serial on the transmit pin (and conversely).

Most of the electronics of the serial port is found in a computer chip (or a part of a chip) known as a UART. For more details on UARTs see the section "What are UARTS" in the Serial−HOWTO.

But you may want to finish this section first so that you will hopefully understand how the UART fits into the overall scheme of things.

Pins and Wires

Old PC's used 25 pin connectors but only about 9 pins were actually used so today most connectors are only 9−pin. Each of the 9 pins usually connects to a wire. Besides the two wires used for transmitting and receiving data, another pin (wire) is signal ground. The voltage on any wire is measured with respect to this ground. Thus the minimum number of wires to use for 2−way transmission of data is 3. Except that it has been known to work with no signal ground wire but with degraded performance and sometimes with errors.

There are still more wires which are for control purposes (signalling) only and not for sending bytes. All of these signals could have been shared on a single wire, but instead, there is a separate dedicated wire for every type of signal. Some (or all) of these control wires are called "modem control lines". Modem control wires are either in the asserted state (on) of +12 volts or in the negated state (off) of −12 volts. One of these wires is to signal the computer to stop sending bytes out the serial port cable. Conversely, another wire signals the device attached to the serial port to stop sending bytes to the computer. If the attached device is a modem, other wires may tell the modem to hang up the telephone line or tell the computer that a connection has been made or that the telephone line is ringing (someone is attempting to call in). See the Serial−HOWTO: Pinout and Signals for more details.

Internal Modem Contains Serial Port

For an internal modem there is no 9−pin connector but the behavior is almost exactly as if the above mentioned cable wires existed. Instead of a a 12 volt signal in a wire giving the state of a modem control line, the internal modem may just use a status bit in its own memory (a register) to determine the state of this non−existent "wire". The internal modem's serial port looks just like a real serial port to the computer. It even includes the speed limits that one may set at ordinary serial ports such as 115200 bits/sec. Unfortunately for Linux, many internal modems today don't work exactly this way but instead use software (running on the CPU) to do much of the modem's work. Unfortunately, such software is often only available for the MS Windows OS (it hasn't been ported to Linux). Thus you can't use most of these modems with Linux See Software−based Modems (winmodems).

4.3 IO Address & IRQ

Since the computer needs to communicate with each serial port, the operating system must know that each serial port exists and where it is (its I/O address). It also needs to know which wire (IRQ number) the serial port must use to request service from the computer's CPU. It requests service by sending an interrupt on this wire. Thus every serial port device must store in its non−volatile memory both its I/O address and its Interrupt ReQuest number: IRQ. See Interrupts. For the PCI bus it doesn't work exactly this way since the PCI bus has its own system of interrupts. But since the PCI−aware BIOS sets up chips to map these PCI interrupts to IRQs, it seemingly behaves just as described above except that sharing of interrupts is allowed (2 or more devices may use the same IRQ number).

I/O addresses are not the same as memory addresses. When an I/O addresses is put onto the computer's address bus, another wire is energized. This both tells main memory to ignore the address and tells all devices which have I/O addresses (such as the serial port) to listen to the address to see if it matches the device's. If the address matches, then the I/O device reads the data on the data bus.

4.4 Names: ttyS0, ttyS1, etc.

The serial ports are named ttyS0, ttyS1, etc. (and usually correspond respectively to COM1, COM2, etc. in DOS/Windows). The /dev directory has a special file for each port. Type "ls /dev/ttyS*" to see them. Just because there may be (for example) a ttyS3 file, doesn't necessarily mean that there exists a physical serial port there.

Which one of these names (ttyS0, ttyS1, etc.) refers to which physical serial port is determined as follows. The serial driver (software) maintains a table showing which I/O address corresponds to which ttyS. This mapping of names (such as ttyS1) to I/O addresses (and IRQ's) may be both set and viewed by the "setserial" command. See What is Setserial. This does `not` set the I/O address and IRQ in the hardware itself (which is set by jumpers or by plug−and−play software). Thus what physical port corresponds to say ttyS1 depends both on what the serial driver thinks (per setserial) and what is set in the hardware. If a mistake has been made, the physical port may not correspond to any name (such as ttyS2) and thus it can't be used. See Serial Port Devices /dev/ttyS2, etc. for more details>

4.5 Interrupts

Bytes come in over the phone line to the modem, are converted from analog to digital by the modem and passed along to the serial port on their way to their destination inside your computer. When the serial port receives a number of bytes (may be set to 1, 4, 8, or 14) into its FIFO buffer, it signals the CPU to fetch them by sending an electrical signal known as an interrupt on a certain wire normally used only by that port. Thus the FIFO waits for a number of bytes and then issues an interrupt.

However, this interrupt will also be sent if there is an unexpected delay while waiting for the next byte to arrive (known as a timeout). Thus if the bytes are being received slowly (such as someone typing on a terminal keyboard) there may be an interrupt issued for every byte received. For some UART chips the rule is like this: If 4 bytes in a row could have been received, but none of these 4 show up, then the port gives up waiting for more bytes and issues an interrupt to fetch the bytes currently in the FIFO. Of course, if the FIFO is empty, no interrupt will be issued.

Each interrupt conductor (inside the computer) has a number (IRQ) and the serial port must know which conductor to use to signal on. For example, ttyS0 normally uses IRQ number 4 known as IRQ4 (or IRQ 4). A list of them and more will be found in "man setserial" (search for "Configuring Serial Ports"). Interrupts are issued whenever the serial port needs to get the CPU's attention. It's important to do this in a timely manner since the buffer inside the serial port can hold only 16 (1 in old serial ports) incoming bytes. If the CPU fails to remove such received bytes promptly, then there will not be any space left for any more incoming bytes and the small buffer may overflow (overrun) resulting in a loss of data bytes.

For an external modem, there is no way (such as flow control) to stop the flow rapidly enough to prevent this. For an internal modem the 16−byte FIFO buffer is on the same card and a good modem will not write to it if it's full. Thus a good internal modem will not overrun the 16−byte buffers but it may need to use Modem−to−Modem Flow Control to prevent the modem itself from being overrun. This is one advantage of an internal modem over an external.

Interrupts are also issued when the serial port has just sent out all 16 of its bytes from its small transmit buffer out the external cable. It then has space for 16 more outgoing bytes. The interrupt is to notify the CPU of that fact so that it may put more bytes in the small transmit buffer to be transmitted. Also, when a modem control line changes state an interrupt is issued.

The buffers mentioned above are all hardware buffers. The serial port also has large buffers in main memory. This will be explained later

Interrupts convey a lot of information but only indirectly. The interrupt itself just tells a chip called the interrupt controller that a certain serial port needs attention. The interrupt controller then signals the CPU. The CPU runs a special program to service the serial port. That program is called an interrupt service routine (part of the serial driver software). It tries to find out what has happened at the serial port and then deals with the problem such a transferring bytes from (or to) the serial port's hardware buffer. This program can easily find out what has happened since the serial port has registers at IO addresses known to the the serial driver software. These registers contain status information about the serial port. The software reads these registers and by inspecting the contents, finds out what has happened and takes appropriate action.

4.6 Data Compression (by the Modem)

Before continuing with the basics of the serial port, one needs to understand about something done by the modem: data compression. In some cases this task is actually done by software run on the computer's CPU but unfortunately at present, such software only works for MS Windows. The discussion here will be for the case where the modem itself does the compression since this is what must happen in order for the modem to work under Linux.

In order to send data faster over the phone line, one may compress (encode it) using a custom encoding scheme which itself depends on the data. The encoded data is smaller than the original (less bytes) and can be sent over the Internet in less time. This process is called "data compression".

If you download files from the Internet, they are likely already compressed and it is not feasible for the modem to try to compress them further. Your modem may sense that what is passing thru has already been compressed and refrain from trying a compress it any more. If you are receiving data which has been compressed by the other modem, your modem will decompress it and create many more bytes than were sent over the phone line. Thus the flow of data from your modem into your computer will be higher than the flow over the phone line to you. The ratio of this flow is called the compression ratio. Compression ratios as high as 4 are possible, but not very likely.

4.7 Error Correction

Similar to data compression, modems may be set to do error correction. While there is some overhead cost involved which slows down the byte/sec flow rate, the fact that error correction strips off start and stop bits actually increases the data byte/sec flow rate.

For the serial port's interface with the external world, each 8−bit byte has 2 extra bits added to it: a start−bit and a stop−bit. Without error correction, these extra start and stop bits usually go right thru the modem and out over the phone lines. But when error correction is enabled, these extra bits are stripped off and the 8−bit bytes are put into packets. This is more efficient and results in higher byte/sec flow in spite of the fact that there are a few more bytes added for packet headers and error correction purposes.

4.8 Data Flow (Speeds)

Data (bytes representing letters, pictures, etc.) flows from your computer to your modem and then out on the telephone line (and conversely). Flow rates (such as 56k (56000) bits/sec) are (incorrectly) called "speed". But almost everyone says "speed" instead of "flow rate". If there were no data compression the flow rate from the computer to the modem would be about the same as the flow rate over the telephone line.

Actually there are two different speeds to consider at your end of the phone line:

- The speed on the phone line itself (DCE speed) modem−to−modem
- The speed from your computer's serial port to your modem (DTE speed)

When you dial out and connect to another modem on the other end of the phone line, your modem often sends you a message like "CONNECT 28800" or "CONNECT 115200". What do these mean? Well, its either the DCE speed or the DTE speed. If it's higher than the advertised modem speed it must be the DTE modem−to−computer speed. This is the case for the 115200 speed shown above. The 28800 must be a DCE (modem−to−modem) speed since the serial port has no such speed. One may configure the modem to report either speed. Some modems report both speeds and report the modem−to−modem speed as (for example): CARRIER 28800.

If you have an internal modem you would not expect that there would be any speed limit on the DTE speed from your modem to your computer since you modem is inside your computer and is almost part of your computer. But there is since the modem contains a dedicated serial port within it.

It's important to understand that the average speed is often less than the specified speed, especially on the short DTE computer−to−modem line. Waits (or idle time) result in a lower average speed. These waits may include long waits of perhaps a second due to Flow Control. At the other extreme there may be very short waits (idle time) of several micro−seconds separating the end of one byte and the start of the next byte. In addition, modems will fallback to lower speeds if the telephone line conditions are less than pristine.

For a discussion of what DTE speed is best to use see section What Speed Should I Use.

4.9 Flow Control

Flow control means the ability to slow done the flow of bytes in a wire. For serial ports this means the ability to stop and then restart the flow without any loss of bytes. Flow control is needed for modems to allow a jump in instantaneous flow rates.

Example of Flow Control

For example, consider the case where you connect a 36.6k external modem via a short cable to your serial port. The modem sends and receives bytes over the phone line at 36.6k bits per second (bps). Assume it's not doing any data compression or error correction. You have set the serial port speed to 115,200 bits/sec (bps), and you are sending data from your computer to the phone line. Then the flow from the your computer to your modem over the short cable is at 115.2k bps. However the flow from your modem out the phone line is only 33.6k bps. Since a faster flow (115.2k) is going into your modem than is coming out of it, the modem is storing the excess flow (115.2k −33.6k = 81.6k bps) in one of its buffers. This buffer would soon overrun (run out of free storage space) unless the high 115.2k flow is stopped.

But now flow control comes to the rescue. When the modem's buffer is almost full, the modem sends a stop signal to the serial port. The serial port passes on the stop signal on to the device driver and the 115.2k bps flow is halted. Then the modem continues to send out data at 33.6k bps drawing on the data it previous accumulated in its buffer. Since nothing is coming into the buffer, the level of bytes in it starts to drop. When almost no bytes are left in the buffer, the modem sends a start signal to the serial port and the 115.2k flow from the computer to the modem resumes. In effect, flow control creates an average flow rate in the short cable (in this case 33.6k) which is significantly less than the "on" flow rate of 115.2k bps. This is "start−stop" flow control.

In the above simple example it was assumed that the modem did no data compression. This would be true when the modem is sending a file which is already compressed and can't be compressed further. Now let's consider the opposite extreme where the modem is compressing the data with a high compression ratio. In such a case the modem might need an input flow rate of say 115.2k bps to provide an output (to the phone line) of 33.6k bps (compressed data). The compression ratio is 3.43 (115.2/33.6) which is much higher than average. In this case the modem is able to compress and the 115.2 bps PC−to−modem flow and send the same data out on the phone line at 33.6bps. There's no need for flow control here. But such a high compression ratio rarely happens so that most of the time flow control is needed to slow down the flow on the 115.2 bps PC−to−modem cable. The flow is stopped and started so that the average flow is usually well under the "on" flow of 115.2 bps.

In the above example the modem was an external modem. But the same situation exists (as of late 2000) for most internal modems. There is still a speed limit on the PC−to−modem speed even though this flow doesn't take place over an external cable. This makes the internal modems compatible with the external modems.

In the above example of flow control the flow was from the computer to a modem. But there is also flow control which is used for the opposite direction of flow: from a modem (or other device) to a computer. Each direction of flow involve 3 buffers: 1. in the modem 2. in the UART chip (called FIFOs) 3. in main memory managed by the serial driver. Flow control protects certain buffers from overflowing. The small UART FIFO buffers are not protected in this way but rely instead on a fast response to the interrupts they issue. FIFO stand for "First In, First Out" which is the way it handles bytes. All the 3 buffers use the FIFO rule but only one of them also uses it as a name. This is the essence of flow control but there are still some more details.

You don't often need flow control in the direction from the modem to a PC. For complex example of a case where it's needed see "Complex Flow Control Example" in the Serial−HOWTO. But if you don't have a high enough speed set between the modem and the computer (serial port speed) then you do need to slow down the flow coming into the modem from the telephone line. To do this your modem must tell the other modem to stop sending. See Modem−to−Modem Flow Control.

Hardware vs. Software Flow Control

If feasible it's best to use "hardware" flow control that uses two dedicated "modem control" wires to send the "stop" and "start" signals. Modern modems almost always use hardware flow control between the modem and the serial port.

Software flow control uses the main receive and transmit wires to send the start and stop signals. It uses the ASCII control characters DC1 (start) and DC3 (stop) for this purpose. They are just inserted into the regular stream of data. Software flow control is not only slower in reacting but also does not allow the sending of binary data unless special precautions are taken. Since binary data will likely contain DC1 and DC3, special means must be taken to distinguish between a DC3 that means a flow control stop and a DC3 that is part of the binary code. Likewise for DC1. To get software flow control to work for binary data requires both modem (hardware) and software support

Symptoms of No Flow Control

Understanding flow−control theory can be of practical use. For example I used my modem to access the Internet and it seemed to work fine. But after a few months I tried to send long files from my PC to an ISP and a huge amount of retries and errors resulted (but eventually Kermit could send a long file after many retries). Receiving in the other direction (from my ISP to me) worked fine. The problem turned out to be a modem with broken flow control. My modem's buffer was overflowing (overrunning) on long outgoing files since no "stop" signal was ever sent to the computer to halt sending to the modem. There was no problem in the direction from the modem to my computer since the capacity (say 115.2k) was always higher than the flow over the telephone line. The fix was to enable flow control by putting into the init string an enable−flow−control command for the modem. (It should have been enabled by default but something was wrong).

Modem−to−Modem Flow Control

This is the flow control of the data sent over the telephone lines between two modems. Practically speaking, it only exists when you have error correction enabled. Actually, even without error correction it's possible to enable software flow control between modems but it may interfere with sending binary data so it's not often used.

4.10 Data Flow Path; Buffers

Much has been explained about this including flow control, a pair of 16−byte FIFO buffers (in the UART), and a pair of larger buffers inside a device connected to the serial port (such as a modem. But there is still another pair of buffers. These are large buffers (perhaps 8k) in main memory also known as serial port buffers. When an application program sends bytes to the serial port (and modem) they first get stashed in the the transmit serial port buffer in main memory. The pair consists of both this transmit buffer and a receive buffer for the opposite direction of byte−flow. Here's an example diagram for the case of browsing the Internet with a browser. Transmit data flow is left to right while receive flow is right to left.

```
application     8k-byte         16-byte        1k-byte        tele-
BROWSER ------- MEMORY -------- UART --------- MODEM -------- phone
program         buffer          buffer         buffer         line
```

The serial device driver takes out say 16 bytes from this transmit buffer, one byte at a time and puts them into the 16−byte transmit buffer in the serial UART for transmission. Once in that transmit buffer, there is no way to stop them from being transmitted. They are then transmitted to the modem which also has a fair sized (say 1k) buffer. When the device driver (on orders from flow control) stops the flow of outgoing bytes from the computer, what it actually stops is the flow of outgoing bytes from the large transmit buffer in main memory. Even after this has happened and the flow to the modem has stopped, an application program may keep sending bytes to the 8k transmit buffer until it becomes fill.

When it gets fill, the application program can't send any more bytes to it (a "write" statement in a C_program blocks) and the application program temporarily stops running and waits until some buffer space becomes available. Thus a flow control "stop" is ultimately able to stop the program that is sending the bytes. Even though this program stops, the computer does not necessarily stop computing. It may switch to running other processes while it's waiting at a flow control stop. The above was a little oversimplified since there is another alternative of having the application program itself do something else while it is waiting to "write".

4.11 Modem Commands

Commands to the modem are sent to it from the communication software over the same conductor as used to send data. The commands are short ASCII strings. Examples are "AT&K3" for enabling hardware flow control (RTS/CTS) between your computer and modem; and "ATDT5393401 for Dialing the number 5393401. Note all commands are prefaced by "AT". Some commands such as enabling flow control help configure the modem. Other commands such as dialing a number actually do something. There are about a hundred or so different possible commands. When your communication software starts running, it first sends an "init" string of commands to the modem to configure it. All commands are sent on the ordinary data line before the modem dials (or receives a call).

Once the modem is connected to another modem (on−line mode), everything that is sent from your computer to your modem goes directly to the other modem and is not interpreted by the modem as a command. There is a way to "escape" from this mode of operation and go back to command mode where everything sent to the modem will be interpreted as a command. The computer just sends "+++" with a specified time spacing before and after it. If this time spacing is correct, the modem reverts to command mode. Another way to do this is by a signal on a certain modem control line.

There are a number of lists of modem commands on the Internet. The section Web Sites has links to a couple of such web−sites. Different models and brands of modems do not use exactly the same set of such commands. So what works for one modem might not work for another. Some common command (not guaranteed to work on all modems) are listed in this HOWTO in the section Modem Configuration

4.12 Serial Driver Module

The device driver for the serial port is the software that operates the serial port. It is now provided as a serial module. From kernel 2.2 on, this module will normally get loaded automatically if it's needed. In earlier kernels, you had to have `kerneld` running in order to do auto−load modules on demand. Otherwise the serial module needed to be explicitly listed in /etc/modules. Before modules became popular with Linux, the serial driver was usually built into the kernel (and sometimes still is). If it's built−in don't let the serial module load or else you will have two serial drivers running at the same time. With 2 drivers there are all sorts of errors including a possible "I/O error" when attempting to open a serial port. Use "lsmod" to see if the module is loaded.

When the serial module is loaded it displays a message on the screen about the existing serial ports (often showing a wrong IRQ). But once the module is used by `setserial` to tell the device driver the (hopefully) correct IRQ then you should see a second display similar to the first but with the correct IRQ, etc. See "Serial Module" in the Serial−HOWTO. See What is Setserial for more info on `setserial`.

---

5. Configuring Overview

If you want to use a modem only for MS Windows/Dos, then you can just install almost any modem and it will work OK. With a Linux PC it's not always this easy unless you use an external modem. All external modems should work OK (even if they are labeled "Plug and Play") But most new internal modems are Plug−and−Play (PnP) and have PnP serial ports. In some cases (depending both on the modem and your version of Linux) The PnP configuring is built into the serial driver so you don't need to do anything. If it's an ISA modem you may need to use the Linux "isapnp" program to configure it (but this is planned to be built into future drivers ??). See the Plug−and−Play−HOWTO and the isapnp docs for more information.

Since each modem has an associated serial port there are two parts to configuring a modem:

- Configuring the modem's serial port: Done only *partly* by the communication program
- Configuring the modem itself: Done by the communication program

Most of the above configuring (but not necessarily most of the effort) is done by the communication program that you use with the modem such as `minicom`, `seyon`, or `wvdial` (for PPP). If you use the modem for dial−in, then the `getty` program which you use to present outsiders with a login−prompt, will help configure. Thus to configure the modem (and much of the serial port) you need to configure the communication program (such as the PPP dialer or `getty`).

Unfortunately the above configuring doesn't do the low−level configuring of the serial port: setting its IO address and IRQ in both the hardware and the driver. If you are lucky, this will happen automatically when you boot Linux. Setting these in the hardware was formerly done by jumpers but today it's done by "Plug−and−Play" software.

But there's a serious problem: Linux (as of early 2001) is not a true Plug−and−Play operating system but the latest serial drivers handle Plug−and−Play for some modems. In other case you may need to use Plug−and−Play tools to set up the configuration although they are not always very user friendly. This may create a difficult problem for you. The next section will go into this in much more detail.

---

6. Configuring the Serial Port

6.1 PCI Bus Support Underway

Although most PCI modems are "winmodems" without a Linux driver (and will not work under Linux), other PCI serial cards (usually modem cards) will often work OK under Linux. Some need no special support in the serial driver and work fine under Linux once setserial is used to configure them. Other PCI cards need special support in the kernel. Some of these cards are supported by kernel 2.4 (and in later versions of the 2.3 series). Kernel 2.2 has no such support. If your modem (or serial port) happens to be supported, then you shouldn't need to do anything to PnP configure it. The new serial driver will read the id number digitally stored on the card to determine how to support the card. It should assign the I/O address to it, determine it's IRQ, etc. So you don't need to use "setserial" for it.

If you have a PCI internal modem which you are convinced is not a winmodem but it will not work because the latest serial driver doesn't support it, you can help in attempting to create a driver for it. To do this you'll need to contact the maintainer of the serial driver, Theodore (Ted) Y. Ts'o. But first check out the modem list site http://www.idir.net/~gromitkc/winmodem.html for the latest info on PCI modems and related topic. You will need to email Ted Ts'o a copy of the output of "lspci −vv" with full information about the model and manufacturer of the PCI modem (or serial port). Then he will try to point you to a test driver which might work for it. You will then need to get it, compile it and possibly recompile your kernel. Then you will test the driver to see if it works OK for you and report the results to Ted Ts'o. If you are willing to do all the above (and this is the latest version of this HOWTO) then email the needed info to him at: mailto:tytso@mit.edu.

PCI modems are not well standardized. Some use main memory for communication with the PC. It you see 8−digit hexadecimal addresses it's not likely to work with Linux. Some require special enabling of the IRQ. The output of "lspci" can help determine if one can be supported. If you see a 4−digit IO port and no long memory address, the modem might work by just telling "setserial" the IO port and the IRQ. Some people have gotten a 3COM 3CP5610 PCI Modem to work that way.

6.2 Configuring Overview

In many cases, configuring will happen automatically and you have nothing to do. But sometimes you need to configure (or just want to check out the configuration). If so, first you need to know about the two parts to configuring the serial port under Linux:

The first part (low−level configuring) is assigning it an IO address, IRQ, and name (such as ttyS2). This IO−IRQ pair must be set in both the hardware and told to the serial driver. We might just call this "io−irq" configuring for short. The `setserial` is used to tell the driver. PnP methods, jumpers, etc, are used to set the hardware. Details will be supplied later. If you need to configure but don't understand certain details it's easy to get into trouble.

The second part (high−level configuring) is assigning it a speed (such as 38.4k bits/sec), selecting flow control, etc. This is often done by communication programs such as PPP, minicom, or by getty (which you may run on the port so that others may log into your computer). However you will need to tell these programs what speed you want, etc. by using a menu or a configuration file. This high−level configuring may also be done with the `stty` program. `stty` is also useful to view the current status if you're having problems. See also the Serial−HOWTO section: "Stty". When Linux starts, some effort is made to detect and configure (low−level) a few serial ports. Exactly what happens depends on your BIOS, hardware, Linux distribution, etc. If the serial ports work OK, there may be no need for you to do any configuring. Application programs often do the high−level configuring but you may need to supply them with the required information. With Plug−and−Play serial ports (often built into an internal modem), the situation has become more complex. Here are cases when you need to do low−level configuring (set IRQ and IO addresses):

- Plan to use more than 2 serial ports
- Installing a new serial port (such as an internal modem)
- Having problems with serial port(s)

For kernel 2.2+ you may be able to use more that 2 serial ports without low−level configuring by sharing interrupts. This only works if the serial hardware supports it and may be no easier than low−level configuring. See Interrupt sharing and Kernels 2.2+

The low−level configuring (setting the IRQ and IO address) seems to cause people more trouble (than high−level), although for many it's fully automatic and there is no configuring to be done. Thus most all of this section is on that topic. Until the serial driver knows the correct IRQ and IO address the port will not work at all. It may not even be found by Linux. Even if it can be found, it may work extremely slow if the IRQ is wrong. See Extremely Slow: Text appears on the screen slowly after long delays.

In the Wintel world, the IO address and IRQ are called "resources" and we are thus configuring certain resources. But there are many other types of "resources" so the term has many other meanings. In review, the low−level configuring consists of putting two values (an IRQ number and IO address) into two places:

1. the device driver (often by running "`setserial`" at boot−time)
2. memory registers of the serial port hardware itself

You may watch the start−up (= boot−time) messages. They are usually correct. But if you're having problems, there's a good chance that some of these messages don't show the true configuration of the hardware (and they are not supposed to). See I/O Address & IRQ: Boot−time messages.

6.3 Common mistakes made re low−level configuring

Here are some common mistakes people make:

- setserial command: They run it (without the "autoconfig" and auto_irq options) and think it has checked out the hardware (it hasn't).
- setserial messages: They see them displayed on the screen at boot−time (or by giving the setserial command) and erroneously think that the result always shows how their hardware is actually configured.
- /proc/interrupts: When their serial device isn't in use they don't see its interrupt there, and erroneously conclude that their serial port can't be found (or doesn't have an interrupt set).
- /proc/ioports and /proc/tty/driver/serial: People think this shows the actual hardware configuration when it only shows about the same info (possibly erroneous) as setserial.

6.4 I/O Address & IRQ: Boot−time messages

In many cases your ports will automatically get low−level configured at boot−time (but not always correctly). To see what is happening, look at the start−up messages on the screen. Don't neglect to check the messages from the BIOS before Linux is loaded (no examples shown here). These BIOS messages may be frozen by pressing the Pause key. Use Shift−PageUp to scroll back to the messages after they have flashed by. Shift−PageDown will scroll in the opposite direction. The `dmesg` command may be used at any time to view some of the messages but it often misses important ones. Here's an example of the start−up messages (as of mid 1999). Note that ttyS00 is the same as /dev/ttyS0.

```
At first you see what was detected (but the irq is only a wild guess):

Serial driver version 4.27 with no serial options enabled
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
ttyS02 at 0x03e8 (irq = 4) is a 16550A

Later you see what was saved, but it's not necessarily correct either:

Loading the saved-state of the serial devices...
/dev/ttyS0 at 0x03f8 (irq = 4) is a 16550A
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
/dev/ttyS2 at 0x03e8 (irq = 5) is a 16550A
```

Note that there is a slight disagreement: The first message shows ttyS2 at irq=4 while the second shows it at irq=5. Your may only have the first message. In most cases the last message is the correct one. But if your having trouble it may be misleading. Before reading the explanation of all of this complexity in the rest of this section, you might just try using your serial port and see if it works OK. If so it may not be essential to read further.

The second message is from the `setserial` program being run at boot−time. It shows what the device driver thinks is the correct configuration. But this too could be wrong. For example, the irq could actually be set to irq=8 in the hardware (both messages wrong). The irq=5 could be there because someone incorrectly put this into a configuration file (or the like). The fact that Linux sometimes gets IRQs wrong is because it doesn't by default probe for IRQs. It just assumes the "standard" ones (first message) or accepts what you told it when you configured it (second message). Neither of these is necessarily correct. If the serial driver has the wrong IRQ the serial port is very slow or doesn't seem to work at all.

The first message is a result of Linux probing the serial ports but it doesn't probe for IRQs. If a port shows up here it exists but the IRQ may be wrong. Linux doesn't check IRQs because doing so is not foolproof. It just assumes the IRQs are as shown because they are the "standard" values. Your may check them manually with `setserial` using the `autoconfig` and `auto_irq` options but this isn't guaranteed to be correct either.

The data shown by the BIOS messages (which you see at first) is what is set in the hardware. If your serial port is Plug−and−Play PnP then it's possible that the `isapnp` will run and change these settings. Look for messages about this after Linux starts. The last serial port message shown in the example above should agree with the BIOS messages (as possibly modified by isapnp). If they don't agree then you either need to change the setting in the port hardware or use setserial to tell the driver what is actually set in the hardware.

Also, if you have Plug−and−Play (PnP) serial ports, Linux will not find them unless the IRQ and IO has been set inside the hardware by Plug−and−Play software. Prior to kernel 2.4 this was a common reason why the start−up messages did not show a serial port that physically exists. The PC hardware (a PnP BIOS) may automatically low−level configure this. PnP configuring will be explained later.

6.5 What is the current IO address and IRQ of my Serial Port ?

If your serial port seems to work OK, then you may type "setserial −g /dev/ttyS*", look at /proc/tty/driver/serial, or inspect the start−up messages. If you serial port doesn't work (or is very slow) then you need to read further.

There are really two answers to the question "What is my IO and IRQ?" 1. What the device driver thinks has been set (This is what setserial usually sets and shows). 2. What is actually set in the hardware. They both should be the same. If they're not it spells trouble since the driver has incorrect info on the physical serial port. If the driver has the wrong IO address it will try to send data to a non−existing serial port −−or even worse, to some other device. If it has the wrong IRQ the driver will not get interrupt service requests from the serial port, resulting in a very slow or no response. See Extremely Slow: Text appears on the screen slowly after long delays. If it has the wrong model of UART there is also apt to be trouble. To determine if both I0−IRQ pairs are identical you must find out how they are set in both the driver and the hardware.

What does the device driver think?

This is easy to find out. Just look at the start−up messages or type "setserial −g /dev/ttyS*". If everything works OK then what it tells you is likely also set in the hardware. There are some other ways to find this info by looking at "files" in the /proc directory. Be warned that there is no guarantee that the same is set in the hardware.

`/proc/ioports` will show the IO addresses that the drivers are using. `/proc/interrupts` shows the IRQs that are used by drivers of currently running processes (that have devices open). It shows how many interrupts have actually be issued. `/proc/tty/driver/serial` shows most of the above, plus the number of bytes that have been received and sent (even if the device is not now open).

Note that for the IO addresses and IRQ assignments, you are only seeing what the driver thinks and not necessarily what is actually set in the hardware. The data on the actual number of interrupts issued and bytes processed is real however. If you see a large number of interrupts and/or bytes then it probably means that the device is (or was in the case of bytes) working. If there are no bytes received (rx:0) but bytes were transmitted (tx:3749 for example), then only one direction of flow is working (or being utilized).

Sometimes a showing of just a few interrupts doesn't mean that the interrupt is actually being physically generated by any serial port. Thus if you see almost no interrupts for a port that you're trying to use, that interrupt might not be set in the hardware and it implies that the driver is using the wrong interrupt. To view /proc/interrupts to check on a program that you're currently running (such as "minicom") you need to keep the program running while you view it.

What is set in my serial port hardware ?

How do you find out what IO address and IRQ are actually set in the device hardware? Perhaps the BIOS messages will tell you some info before Linux starts booting. Use the shift−PageUp key to step back thru the boot−time messages and look at the very first ones which are from the BIOS. This is how it was before Linux started. Setserial can't change it but isapnp or pciutils can and starting with kernel 2.4, these will be built into the serial driver.

One crude method is try probing with setserial using the "autoconfig" option. You'll need to guess the addresses to probe at. See What is Setserial. For a PCI serial port, use the "lspci" command (for kernels <2.2 look at /proc/pci). If your serial port is is Plug−and−Play see the next two subsections.

For a port set with jumpers, its how the jumpers were set. If the port is not Plug−and−Play (PnP) but has been setup by using a DOS program then it's set at whatever the person who ran that program set it to.

What is set in my PnP serial port hardware ?

PnP ports don't store their configuration in the hardware when the power is turned off. This is in contrast to Jumpers (non−PnP) which remain the same with the power off. If you have an ISA PnP port, it can reach a state where it doesn't have any IO address or IRQ and is in effect disabled. It should still be possible to find the port using the `pnpdump` program.

For Plug−and−Play (PnP) on the ISA bus one may try the `pnpdump` program (part of `isapnptools`). If you use the −−dumpregs option then it should tell you the actual IO address and IRQ set in the port. The address it "trys" is not the device's IO address, but a special

For PnP ports checking on how it's configured under DOS/Windows may not be of much help. Windows stores its configuration info in its Registry which is not used by Linux. It may supply the BIOS's non−volatile memory with some info but it may not be kept in sync with the current Window configuration in the Registry ?? If you let a PnP BIOS automatically do the configuring when you start Linux (and have told the BIOS that you don't have a PnP operating system when running Linux) then Linux should use whatever configuration is in the BIOS's non−volatile memory.

6.6 Choosing Serial IRQs

If you have Plug−and−Play ports then either a PnP BIOS or the serial driver may configure all your devices for you and then you may not need to choose any IRQs. PnP determines what it thinks is best and assigns them. But if you use the tools in Linux for Plug−and−Play (isapnp and pcitools) or jumpers then you have to choose. If you already know what IRQ you want to use you could skip this section except that you may want to know that IRQ 0 has a special use (see the following paragraph).

IRQ 0 is not an IRQ

While IRQ 0 is actually the timer (in hardware) it has a special meaning for setting a serial port with setserial. It tells the driver that there is no interrupt for the port and the driver then will use polling methods. This is quite inefficient but can be tried if there is an interrupt conflict or mis−set interrupt. The advantage of assigning this is that you don't need to know what interrupt is set in the hardware. It should be used only as a temporary expedient until you are able to find a real interrupt to use.

Interrupt sharing and Kernels 2.2+

The general rule is that every device should use a unique IRQ and not share them. But there are situations where sharing is permitted such as with most multi−port boards. Even when it is permitted, it may not be as efficient since every time a shared interrupt is given a check must be made to determine where it came from. Thus if it's feasible, it's nice to allocate every device its own interrupt.

Prior to kernel 2.2, serial IRQs could be shared with each other only for most multiport boards. Starting with kernel 2.2 serial IRQs may be sometimes shared between all serial ports. In order for sharing to work in 2.2 the kernel must have been compiled with CONFIG_SERIAL_SHARE_IRQ, and the serial port hardware must support sharing (so that if two serial cards put different voltages on the same interrupt wire, only the voltage that means "this is an interrupt" will prevail). Thus even if you have 2.2, it may be best to avoid sharing.

What IRQs to choose?

The serial hardware often has only a limited number of IRQs it can be set at. Also you don't want IRQ conflicts. So there may not be much of a choice. Your PC may normally come with `ttyS0` and `ttyS2` at IRQ 4, and `ttyS1` and `ttyS3` at IRQ 3. Looking at `/proc/interrupts` will show which IRQs are being used by programs currently running. You likely don't want to use one of these. Before IRQ 5 was used for sound cards, it was often used for a serial port.

Here is how Greg (original author of Serial−HOWTO) set his up in /etc/rc.d/rc.serial. rc.serial is a file (shell script) which runs at start−up (it may have a different name of location). For versions of "setserial" after 2.15 it's not always done this way anymore but this example does show the choice of IRQs.

```
/sbin/setserial /dev/ttyS0 irq 3        # my serial mouse
/sbin/setserial /dev/ttyS1 irq 4        # my Wyse dumb terminal
/sbin/setserial /dev/ttyS2 irq 5        # my Zoom modem
/sbin/setserial /dev/ttyS3 irq 9        # my USR modem
```

Standard IRQ assignments:

```
 IRQ  0    Timer channel 0 (May mean "no interrupt".  See below.)
 IRQ  1    Keyboard
 IRQ  2    Cascade for controller 2
 IRQ  3    Serial port 2
 IRQ  4    Serial port 1
 IRQ  5    Parallel port 2, Sound card
 IRQ  6    Floppy diskette
 IRQ  7    Parallel port 1
 IRQ  8    Real-time clock
 IRQ  9    Redirected to IRQ2
 IRQ 10    not assigned
 IRQ 11    not assigned
```

```
IRQ 12    not assigned
IRQ 13    Math coprocessor
IRQ 14    Hard disk controller 1
IRQ 15    Hard disk controller 2
```

There is really no Right Thing to do when choosing interrupts. Just make sure it isn't being used by the motherboard, or any other boards. 2, 3, 4, 5, 7, 10, 11, 12 or 15 are possible choices. Note that IRQ 2 is the same as IRQ 9. You can call it either 2 or 9, the serial driver is very understanding. If you have a very old serial board it may not be able to use IRQs 8 and above.

Make sure you don't use IRQs 1, 6, 8, 13 or 14! These are used by your motherboard. You will make her very unhappy by taking her IRQs. When you are done, double−check `/proc/interrupts` when programs that use interrupts are being run and make sure there are no conflicts.

6.7 Choosing Addresses −−Video card conflict with ttyS3

The IO address of the IBM 8514 video board (and others like it) is allegedly 0x?2e8 where ? is 2, 4, 8, or 9. This may conflict with the IO address of `ttyS3` at 0x02e8. Your may think that this shouldn't happen since the addresses are different in the high order digit (the leading 0 in 02e8). You're right, but a poorly designed serial port may ignore the high order digit and respond to any address that ends in 2e8. That is bad news if you try to use `ttyS3` (ISA bus) at this IO address.

For the ISA bus you should try to use the default addresses shown below. Addresses shown represent the first address of an 8−byte range. For example 3f8 is really the range 3f8−3ff. Each serial device (as well as other types of devices that use IO addresses) needs its own unique address range. There should be no overlaps (conflicts). Here are the default addresses for commonly used serial ports on the ISA bus:

```
ttyS0 address 0x3f8
ttyS1 address 0x2f8
ttyS2 address 0x3e8
ttyS3 address 0x2e8
```

Suppose there is an address conflict (as reported by `setserial −g /dev/ttyS*`) between a real serial port and another port which does not physically exist (and shows UART: unknown). Such a conflict shouldn't cause problems but it sometimes does in older kernels. To avoid this problem don't permit such address conflicts or delete /dev/ttySx if it doesn't physically exist.

6.8 Set IO Address & IRQ in the hardware (mostly for PnP)

After it's set in the hardware don't forget to insure that it also gets set in the driver by using `setserial`. For non−PnP serial ports they are either set in hardware by jumpers or by running a DOS program ("jumperless") to set them (it may disable PnP). The rest of this subsection is only for PnP serial ports. Here's a list of the possible methods of configuring PnP serial ports:

- Using a PnP BIOS CMOS setup menu (usually only for external modems on ttyS0 (Com1) and ttyS1 (Com2))
- Letting a PnP BIOS automatically configure a PnP serial port See Using a PnP BIOS to I0−IRQ Configure
- Doing nothing if you have both a PnP serial port and a PnP Linux operating system (see Plug−and−Play−HOWTO).
- Using `isapnp` for a PnP serial port non−PCI)

* Using pciutils (pcitools) for the PCI bus

The IO address and IRQ must be set (by PnP) in their registers each time the system is powered on since PnP hardware doesn't remember how it was set when the power is shut off. A simple way to do this is to let a PnP BIOS know that you don't have a PnP OS and the BIOS will automatically do this each time you start. This might cause problems in Windows (which is a PnP OS) if you start Windows with the BIOS thinking that Windows is not a PnP OS. See Plug−and−Play−HOWTO.

Plug−and−Play was designed to automate this io−irq configuring, but for Linux at present, it has made life more complicated. The standard kernels for Linux don't support plug−and−play very well. If you use a patch to the Linux kernel to covert it to a plug−and−play operating system, then all of the above should be handled automatically by the OS. But when you want to use this to automate configuring devices other that the serial port, you may find that you'll still have to configure the drivers manually since many Linux drivers are not written to support a Linux PnP OS. If you use `isapnptools` or the BIOS for configuring plug−and−play this will only put the two values into the registers of the serial port section of the modem card and you will likely still need to set up setserial. None of this is easy or very well documented as of early 1999. See Plug−and−Play−HOWTO and the isapnptools FAQ.

Using a PnP BIOS to I0−IRQ Configure

While the explanation of how to use a PnP OS or isapnp for io−irq configuring should come with such software, this is not the case if you want to let a PnP BIOS do such configuring. Not all PnP BIOS can do this. The BIOS usually has a CMOS menu for setting up the first two serial ports. This menu may be hard to find and for an "Award" BIOS it was found under "chipset features setup" There is often little to choose from. Unless otherwise indicated in a menu, these first two ports normally get set at the standard IO addresses and IRQs. See Serial Port Device Names & Numbers

Whether you like it or not, when you start up a PC a PnP BIOS starts to do PnP (io−irq) configuring of hardware devices. It may do the job partially and turn the rest over to a PnP OS (which you probably don't have) or if thinks you don't have a PnP OS it may fully configure all the PnP devices but not configure the device drivers. This is what you want but it's not always easy to figure out exactly what the PnP BIOS has done.

If you tell the BIOS that you don't have a PnP OS, then the PnP BIOS should do the configuring of all PnP serial ports −−not just the first two. An indirect way to control what the BIOS does (if you have Windows 9x on the same PC) is to "force" a configuration under Windows. See Plug−and−Play−HOWTO and search for "forced". It's easier to use the CMOS BIOS menu which may override what you "forced" under Windows. There could be a BIOS option that can set or disable this "override" capability.

If you add a new PnP device, the BIOS should change its PnP configuration to accommodate it. It could even change the io−irq of existing devices if required to avoid any conflicts. For this purpose, it keeps a list of non−PnP devices provided that you have told the BIOS how these non−PnP devices are io−irq configured. One way to tell the BIOS this is by running a program called ICU under DOS/Windows.

But how do you find out what the BIOS has done so that you set up the device drivers with this info? The BIOS itself may provide some info, either in its setup menus of via messages on the screen when you turn on your computer. See What is set in my serial port hardware?

6.9 Giving the IRQ and IO Address to Setserial

Once you've set the IRQ and IO address in the hardware (or arranged for it to be done by PnP) you also need to insure that the "setserial" command is run each time you start Linux. See the subsection Boot−time Configuration

6.10 Other Configuring

Configuring Hardware Flow Control (RTS/CTS)

See Flow Control for an explanation of it. You should always use hardware flow control if possible. Your communication program or "`getty`" should have an option for setting it (and if you're in luck it might be enabled by default). It needs to be set both inside your modem (by an init string or default) and in the device driver. Your communication program should set both of these (if you configure it right).

If none of the above will fully enable hardware flow control. Then you must do it yourself. For the modem, make sure that it's either done by the init string or is on by default. If you need to tell the device driver to do it is best done on startup by putting it in a file that runs at boot−time. See the subsection Boot−time Configuration You need to add the following to such a file for each serial port (example is ttyS2) you want to enable hardware flow control on:

```
stty crtscts < /dev/ttyS2
or
stty −F /dev/ttyS2 crtscts
```

If you want to see if flow control is enabled do the following: In minicom (or the like) type AT&V to see how the modem is configured and look for &K3 which means hardware flow control. Then see if the device driver knows about it by typing: stty −F /dev/ttyS2 −a Look for "crtscts" (without a disabling minus sign).

---

7. Modem Configuration (excluding serial port)

7.1 Finding Your Modem

Before spending a lot of time deciding how to configure your modem, you first need to make sure it can be found and that AT−commands and the like can be sent to it. So I suggest you first give it a very simple configuration using the communication program you will be using on the port and see it it works. If this works you may then want to improve on the configuration, If not then see My Modem is Physically There but Can't be Found. A winmodem may be hard to find and will not work under Linux.

7.2 AT Commands

While the serial port on which a modem resides requires configuring, so does the modem itself. The modem is configured by sending AT commands (or the like) to it on the same serial line that is used to send data.

Most modems use an AT command set. These are cryptic and short ASCII commands where all command strings must be prefaced by the letters AT. AT means: ATtention, expect a command to follow. For example:

ATZ&K3<return> This is an AT−command string with two commands here: Z and &K3. Z is short for Z0 and a few modems require that you use Z0 instead of just Z. It's like this for other commands ending in 0. The command string is terminated by a return character (use the <enter> key if you are manually typing it). A string that's too long (40 or more characters) may not work on older modems. You may use either uppercase or lowercase letters.

Unfortunately there are many different variations of the AT command set so that what works for one modem may or may not work for another modem. Thus there is no guarantee that the AT commands given in this section will work on your modem.

Such command strings are either automatically sent to the modem by communication programs or are manually typed in by you. Most communication programs provide a screen where you may change (edit) and save the init string that the communication program will use. The modem itself has a stored configuration (profile) which is like a long init string. It represents the configuration of the modem when it's first tuned on. You may change it to suit your taste. In most cases there are a few different such configurations (profiles) and there are ways to designate one of them to be active.

If you have a manual for your modem (either on paper or on floppy disk) you might find AT−commands there. 3Com modems (and others ??) have AT−Command help files built into the modem so if you type say "AT$" to the modem it will display some "online help".

You can also find info on AT commands on the Internet. You should first try a site for your modem manufacturer. If this doesn't work out then you can search the Internet using terms that are from AT commands such as &C1, &D3, etc. This will tend to find sites that actually list AT−Commands instead of sites that just talk about them in general. You might also try a few of the sites listed in the subsection Web Sites. Be warned that the AT−commands for a different brand of modem may be somewhat different.

7.3 Init Strings: Saving and Recalling

The examples given in this subsection are from the Hayes AT modem command set. All command strings must be prefaced by the two letters AT. For example: AT&C1&D3^M (^M is the return character). When a modem is powered on, it automatically configures itself with one of the configurations it has stored in its non−volatile memory. If this configuration is satisfactory there is nothing further to do.

If it's not satisfactory, then one may either alter the stored configuration or configure the modem each time you use it by sending it a string of commands known as an "init string" (= initialization string). Normally, a communication program does this. What it sends will depend on how you configured the communications program. Your communication program should allow you to edit the init string and change it to whatever you want. Sometimes the communications program will let you select the model of your modem and then it will use an init string that it thinks is best for that modem.

The configuration of the modem when it's first powered on may be expressed by an init string. You might think of this as the default "string" (called a profile). If your communications program sends the modem another string (the init string), then this string will modify the default configuration. For example, if the init string only contains two commands, then only those two items will be changed. However, some commands will recall a stored profile from inside the modem so a single such command in the init string can thereby change everything in the configuration.

Modern modems have a few different stored profiles to choose from that are stored in the modem's non−volatile memory (it's still there when you turn it off). In my modem there are two factory profiles (0 and

1, neither of which you can change) and two user defined profiles (0 and 1) that the user may set and store. Your modem may have more. To view some of these profiles send the command &V. At power−up one of the user−defined profiles is loaded. For example, if you type the command &Y0 (just Y0 for a 3Com modem) then in the future profile 0 will be used at power−on.

There are also commands to load (activate) any of the stored profiles. Such a load command may be put in an init string. Of course if it loads the same profile that was automatically loaded at power−up, nothing is changed (unless the active profile has been modified since power−up). Since your profile could have thus been modified it's a good idea to use some kind of an init string even if it does nothing more than load a stored profile.

Examples of loading saved profiles:
Z0 loads user−defined profile 0 and resets (hangs up, etc.)
&F1 loads factory profile 1

Once you have sent commands to the modem to configure it the way you want (such as loading a factory profile and modifying it a little) you may save this as a user−defined profile:
&W0 saves the current configuration to user−profile 0.

Many people don't bother saving a good configuration in their modem, but instead, send the modem a longer init string each time the modem is used. Another method is to restore the factory default by &F1 at the start of the init string and then modify it a little by adding a few other commands to the end of the init string. Since there is no way to modify the factory default this prevents anyone from changing the configuration by modifying (and saving) the user−defined profile.

You may choose an init string supplied by someone else that they think is right for your modem. Some communication programs have a library of init strings to select from. The most difficult method (and one which will teach you the most about modems) is to study the modem manual and write one yourself. You could save this configuration inside the modem so that you don't need an init string. A third alternative is to start with an init string that someone else wrote, but modify it to suit your purposes.

If you look at init strings used by communication programs you may see symbols which are not valid modem commands. These symbols are commands to the communication program itself and will not be sent to the modem. For example, may mean to pause briefly.

Where is my "init string" so I can modify it ?

This depends on your communication program (often a PPP program). If this is the latest version of Modem−HOWTO send me info for other cases.

- Gnome: run pppsetup
- wvdial: edit /etc/wvdial.conf
- minicom: hit ^Ao (or possibly ALT−o), then select "Modem and Dialing"

7.4 Other AT Modem Commands

For dial−in see Dial−in Modem Configuration. The rest of this section is mostly what was in the old Serial−HOWTO. All strings must start with AT. Here's a few Hayes AT codes that should be in the string (if they are not set by using a factory default or by a saved configuration).

- E1 command echo ON
- Q0 result codes are reported
- V1 result codes are verbose
- S0=0 never answer (uugetty does this with the WAITFOR option)

Here's some more AT commands for special purposes:

- &C1 CD is only on when you're connected
- &S0 DSR is always on
- &X3 Dial even if there is no dialtone (Use where dial−tones don't exist).

Greg Hankins had a collection of setups for different types of modems. It's not currently maintained and covers modems prior to 1998.
`ftp://ftp.cc.gatech.edu/pub/people/gregh/modem−configs`.

Note: to get his USR Courier V.34 modem to reset correctly when DTR drops, Greg Hankins had to set `&D2` and `S13=1` (this sets bit 0 of register S13). This has been confirmed to work on USR Sportster V.34 modems as well.

Note: some Supra modems treat CD differently than other modems. If you are using a Supra, try setting `&C0` and *not* `&C1`. You must also set `&D2` to handle DTR correctly.

7.5 Blacklisting

If phone number is dialed a few times with no success, some modems may blacklist a phone number. After a certain time you may try again. Some countries require this to reduce needless repeated dialing. To view the blacklist try %B. To delete the blacklist use these AT commands:

- SR Robotics o 3COM: s40=2 or if NG try s40=7
- Lucent: %t21,18,0
- Rockwell: %tcb
- Cirrus Logic: *nc9

7.6 What AT Commands are Now Set in my Modem?

You may try to use minicom for viewing your modem profile. It's best not to have any other process running on the modem port when you do this. If you have set up minicom for your modem, then you may type on the command line: `minicom −o` to start minicom without restoring the saved modem profile. Then type `at&v` to display the profile. To exit minicom without disturbing this profile, use the q (quit) command for exiting without resetting.

The above may not work for various reasons. If the modem has been set not to echo result codes it may not even display any profile. If there is another process running on the modem port at the same time, some of what the modem sends to you is likely to be read by the other process so you will see only part of the profile. Is there some way to temporarily stop the other process on the port so it will not interfere? I tried the "stop" signal using the "kill" command but it didn't work. If this is the latest version of this HOWTO, let me know if you find a way to do it.

If you have at least one process running on the modem port and kill them, the modem's profile may be reset so you will not observe what the original profile was. This will happen if you kill getty (or it's replacements:

login or bash) and have &D3 set. The killing of getty (or the like) will drop DTR and reset the modem's profile to the power−on state. To keep getty from respawning when killed, comment it out in /etc/inittab and do an "init q".

7.7 Modem States (or Modes)

Since the channel for sending AT commands to the modem is the same channel that is used for the flow of data (files, packets, etc.) then it's important to cleanly separate the AT commands from the data.

When the modem is first turned on it's in the command mode (also called terminal mode, idle state or AT−command mode). Anything sent to it from the PC is assumed to be an AT command and not data. Then if a dial command is sent to it (ATD...), it dials and connects to another modem. It's now in the on−line data mode (connected) and sends and receives data (such as Internet pages). In this mode AT command one trys to send it will not work but will be transmitted to the other modem instead. Except for the escape command. This is +++ with a minimum time delay both at the start and end. The time delay allows the modem to determine that it is likely a real escape and not just +++ in a file being transmitted.

So we have two states so far: AT−command and on−line data. But there is a third important state which is sort of a combination of these two. It's the on−line command mode. This is when the modem maintains a connection (without sending/receiving data) but anything sent from the PC is interpreted as an AT command. This is the state reached with a +++ escape signal or by a DTR drop from the PC provided the &D1 has been set. Then one can send AT commands to the modem including commands which will leave this state and go to one of the other two states.

There are other states also: dialing state and handshaking state but they normally lead to the connected (on−line) state. If they don't then the modem should hang up, thereby returning to the initial AT−command (or idle) state.

---

8. Serial Port Devices /dev/ttyS2, etc.

For creating devices in the device directory see: the Serial−HOWTO: "Creating Devices In the /dev directory".

8.1 Devfs (The new Device File System)

This is a new type of device interface to Linix. It's optional starting with kernel 2.4. It's more efficient than the conventional interface and makes it easy to deal with a huge number of devices. The device names have all changed as well. But there's an option to continue using the old names. For a detailed description of it see: http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html

The name changes (if used) are: ttyS2 becomes tts/2 (Serial port), tty3 becomes vc/3 (Virtual Console), ptyp1 becomes pty/m1 (PTY master), ttyp2 becomes pty/s2 (PTY slave). "tts" looks like a directory which contains devices "files": 0, 1, 2, etc. All of these new names should still be in the /dev directory although optionally one may put them elsewhere.

8.2 Serial Port Device Names & Numbers

Devices in Linux have major and minor numbers (unless you use the new devfs). The serial port ttySx (x=0,1,2, etc.) has major number 4. You may see this (and the minor numbers too) by typing: "ls −l ttyS*" in the /dev directory.

There formerly was an alternate name for each serial port. For example, ttyS2 would have cua2 as an alternate name. You may still have the cua devices in your /dev directory but they are now deprecated. Their drivers behave slightly different than for the ttyS ones. Device Obsolete">."')

Dos/Windows use the COM name while the `setserial` program uses tty00, tty01, etc. Don't confuse these with dev/tty0, dev/tty1, etc. which are used for the console (your PC monitor) but are not serial ports. The table below is for the "standard" case (but yours could be different).

```
                                                     IO      devfs
 dos              major minor        major minor  address    name
COM1  /dev/ttyS0  4,  64;  /dev/cua0  5,  64      3F8    /dev/tts/0
COM2  /dev/ttyS1  4,  65;  /dev/cua1  5,  65      2F8    /dev/tts/1
COM3  /dev/ttyS2  4,  66;  /dev/cua2  5,  66      3E8    /dev/tts/2
COM4  /dev/ttyS3  4,  67;  /dev/cua3  5,  67      2E8    /dev/tts/3
```

8.3 Universal Serial Bus Ports

Although the USB is not covered in this HOWTO, the serial ports on the USB are: /dev/ttyUSB0 /dev/ttyUSB1, etc.

8.4 Link ttySN to /dev/modem

On some installations, two extra devices will be created, `/dev/modem` for your modem and `/dev/mouse` for a mouse. Both of these are symbolic links to the appropriate serial device in `/dev` which you specified during the installation Except if you have a bus mouse, then `/dev/mouse` will point to the bus mouse device).

Formerly the use of `/dev/modem` was discouraged since lock files might not realize that it was really say `/dev/ttyS2`. The newer lock file system doesn't fall into this trap so it's now OK to use such links.

8.5 cua Device Obsolete

Each ttyS device has a corresponding cua device. But the cua device is deprecated so it's best to use ttyS (unless cua is required). There is a difference between cua and ttyS but a savvy programmer can make a ttyS port behave just like a cua port so there is no real need for the cua anymore. Except that some older programs may need to use the cua.

What's the difference? The main difference between cua and ttyS has to do with what happens in a C−program when an ordinary "open" command tries to open the port. If a cua port has been set to check modem control signals, the port can be opened even if the CD modem control signal says not to. Astute programming (by adding additional lines to the program) can force a ttyS port to behave this way also. But a cua port can be more easily programmed to open for dialing out on a modem even when the modem fails to assert CD (since no one has called into it and there's no carrier). That's why cua was once used for dial−out and ttyS used for dial−in.

Starting with Linux kernel 2.2, a warning message is put in the kernel log when one uses cua. This is an omen that cua is defunct and should be avoided if possible.

9. Interesting Programs You Should Know About

9.1 What is setserial ?

This part is in 3 HOWTOs: Modem, Serial, and Text−Terminal. There are some minor differences, depending on which HOWTO it appears in.

Introduction

Don't ever use `setserial` with Laptops (PCMCIA). `setserial` is a program which allows you to tell the device driver software the I/O address of the serial port, which interrupt (IRQ) is set in the port's hardware, what type of UART you have, etc. Since theres a good chance that the serial ports will be automatically detected and set, many people never need to use `setserial`. In any case setserial will not work without either serial support built into the kernel or loaded as a module. The module may get loaded automatically if you (or a script) tries to use setserial.

Setserial can also show how the driver is currently set. In addition, it can be made to probe the hardware and try to determine the UART type and IRQ, but this has severe limitations. See Probing. Note that it can't set the IRQ or the port address in the hardware of PnP serial ports (but the plug−and−play features of the serial driver may do this).

If you only have one or two built−in serial ports, they will usually get set up correctly without using setserial. Otherwise, if you add more serial ports (such as a modem card) you will likely need to deal with setserial. Besides the man page for `setserial`, check out info in `/usr/doc/setserial.../` or `/usr/share/doc/setserial`. It should tell you how setserial is handled in your distribution of Linux.

`Setserial` is often run automatically at boot−time by a start−up shell−script for the purpose of assigning IRQs, etc. to the driver. Setserial will only work if the serial module is loaded (or if the equivalent was compiled into your kernel). If the serial module gets unloaded later on, the changes previously made by `setserial` will be forgotten by the kernel. But recent (2000) distributions may contain scripts that save and restore this. If not, then `setserial` must be run again to reestablish them. In addition to running via a start−up script, something akin to `setserial` also runs earlier when the serial module is loaded (or the like). Thus when you watch the start−up messages on the screen it may look like it ran twice, and in fact it has.

Setserial does not set either IRQ's nor I/O addresses in the serial port hardware itself. That is done either by jumpers or by plug−and−play. You must tell setserial the identical values that have been set in the hardware. Do not just invent some values that you think would be nice to use and then tell them to setserial. However, if you know the I/O address but don't know the IRQ you may command setserial to attempt to determine the IRQ.

You can see a list of possible commands by just typing `setserial` with no arguments. This fails to show you the one−letter options such as −v for verbose which you should normally use when troubleshooting. Note that setserial calls an IO address a "port". If you type:

```
setserial −g /dev/ttyS*
```

you'll see some info about how that device driver is configured for your ports. Note that where it says `"UART: unknown"` it probably means that no uart exists. In other words you probably have no such serial port and the other info shown about the port is meaningless and should be ignored. If you really do have such a serial port, setserial doesn't recognize it and that needs to be fixed.

If you add −a to the option −g you will see more info although few people need to deal with (or understand) this additional info since the default settings you see usually work fine. In normal cases the hardware is set up the same way as "setserial" reports, but if you are having problems there is a good chance that "setserial" has it wrong. In fact, you can run "setserial" and assign a purely fictitious I/O port address, any IRQ, and whatever uart type you would like to have. Then the next time you type "setserial ..." it will display these bogus values without complaint. They will also be officially registered with the kernel as seen by the "scanport" command. Of course the serial port driver will not work correctly (if at all) if you attempt to use such a port. Thus when giving parameters to "setserial" anything goes. Well almost. If you assign one port a base address that is already assigned (such as 3e8) it will not accept it. But if you use 3e9 it will accept it. Unfortunately 3e9 is already assigned since it is within the range starting at base address 3e8. Thus the moral of the story is to make sure of your data before assigning resources with setserial.

While assignments made by setserial are lost when the PC is powered off, a configuration file may restore them (or a previous configuration) when the PC is started up again. In newer versions, what you change by setserial gets automatically saved to a configuration file. In older versions, the configuration file only changes if you edit it manually so the configuration remains the same from boot to boot. See Configuration Scripts/Files

Probing

With appropriate options, `setserial` can probe (at a given I/O address) for a serial port but you must guess the I/O address. If you ask it to probe for /dev/ttyS2 for example, it will only probe at the address it thinks ttyS2 is at (2F8). If you tell setserial that ttyS2 is at a different address, then it will probe at that address, etc. See Probing

The purpose of this is to see if there is a uart there, and if so, what its IRQ is. Use "setserial" mainly as a last resort as there are faster ways to attempt it such as wvdialconf to detect modems, looking at very early boot−time messages, or using `pnpdump --dumpregs`. To try to detect the physical hardware use the −v (verbose) and `autoconfig` command to `setserial`. If the resulting message shows a uart type such as 16550A, then you're OK. If instead it shows "`unknown`" for the uart type, then there is supposedly no serial port at all at that I/O address. Some cheap serial ports don't identify themselves correctly so if you see "`unknown`" you still might have a serial port there.

Besides auto−probing for a uart type, setserial can auto−probe for IRQ's but this doesn't always work right either. In one case it first gave the wrong irq but when the command was repeated it found the correct irq. In versions of setserial >= 2.15, the results of your last probe test may be saved and put into the configuration file `/etc/serial.conf` which will be used next time you start Linux. At boot−time when the serial module loads (or the like), a probe for UARTs is made automatically and reported on the screen. But the IRQs shown may be wrong. The second report of the same is the result of a script which usually does no probing and thus provides no reliable information as to how the hardware is actually set. It only shows configuration data someone wrote into the script or data that got saved in /etc/serial.conf.

It may be that two serial ports both have the same IO address set in the hardware. Of course this is not permitted but it sometimes happens anyway. Probing detects one serial port when actually there are two.

However if they have different IRQs, then the probe for IRQs may show IRQ = 0. For me it only did this if I first used `setserial` to give the IRQ a ficticious value.

Boot−time Configuration

When the kernel loads the serial module (or if the "module equivalent" is built into the kernel) then only `ttyS{0-3}` are auto−detected and the driver is set to use only IRQs 4 and 3 (regardless of what IRQs are actually set in the hardware). You see this as a boot−time message just like as if `setserial` had been run.

To correct possible errors in IRQs (or for other reasons) there may be a file somewhere that runs `setserial` again. Unfortunately, if this file has some IRQs wrong, the kernel will still have incorrect info about the IRQs. This file should run early at boot−time before any process uses the serial port. In fact, your distribution may have set things up so that the setserial program runs automatically from a start−up script at boot−time. More info about how to handle this situation for your particular distribution might be found in file named "setserial..." or the like located in directory /usr/doc/ or /usr/share/doc/.

Before modifying a configuration file, you can test out a "proposed" `setserial` command by just typing it on the command line. In some cases the results of this use of `setserial` will automatically get saved in /etc/serial.conf when you shutdown. So if it worked OK (and solved your problem) then there's no need to modify any configuration file. See New configuration method using /etc/serial.conf.

Configuration Scripts/Files

Your objective is to modify (or create) a script file in the /etc tree that runs setserial at boot−time. Most distributions provide such a file (but it may not initially reside in the /etc tree). In addition, setserial 2.15 and higher often have an /etc/serial.conf file that is used by the above script so that you don't need to directly edit the script that runs setserial. In addition just using setserial on the command line (2.15+) may ultimately alter this configuration file.

So prior to version 2.15 all you do is edit a script. After 2.15 you may need to either do one of three things: 1. edit a script. 2. edit `/etc/serial.conf` or 3. run "setserial" on the command line which may result in `/etc/serial.conf` automatically being edited. Which one of these you need to do depends on both your particular distribution, and how you have set it up.

Edit a script (required prior to version 2.15)

Prior to setserial 2.15 (1999) there was no /etc/serial.conf file to configure setserial. Thus you need to find the file that runs "setserial" at boot time and edit it. If it doesn't exist, you need to create one (or place the commands in a file that runs early at boot−time). If such a file is currently being used it's likely somewhere in the /etc directory−tree. But Redhat <6.0 has supplied it in /usr/doc/setserial/ but you need to move it to the /etc tree before using it. You might use "locate" to try to find such a file. For example, you could type: locate "*serial*".

The script `/etc/rc.d/rc.serial` was commonly used in the past. The Debian distribution used `/etc/rc.boot/0setserial`. Another file once used was `/etc/rc.d/rc.local` but it's not a good idea to use this since it may not be run early enough. It's been reported that other processes may try to open the serial port before rc.local runs resulting in serial communication failure. Today it's most likely in /etc/init.d/ but it isn't normally intended to be edited.

If such a file is supplied, it should contain a number of commented−out examples. By uncommenting some of these and/or modifying them, you should be able to set things up correctly. Make sure that you are using a valid path for `setserial`, and a valid device name. You could do a test by executing this file manually (just type its name as the super−user) to see if it works right. Testing like this is a lot faster than doing repeated reboots to get it right.

For versions >= 2.15 (provided your distribution implemented the change, Redhat didn't) it may be more tricky to do since the file that runs setserial on startup, /etc/init.d/setserial or the like was not intended to be edited by the user. See New configuration method using /etc/serial.conf.

If you want setserial to automatically determine the uart and the IRQ for ttyS3 you would add something like:

```
/sbin/setserial  /dev/ttyS3 auto_irq skip_test autoconfig
```

Do this for every serial port you want to auto configure. Be sure to give a device name that really does exist on your machine. In some cases this will not work right due to the hardware. If you know what the uart and irq actually are, you may want to assign them explicitly with "setserial". For example:

```
/sbin/setserial /dev/ttyS3 irq 5 uart 16550A  skip_test
```

New configuration method using /etc/serial.conf

Prior to setserial version 2.15, the way to configure setserial was to manually edit the shell−script that ran setserial at boot−time. See Edit a script (after version 2.15: perhaps not). Starting with version 2.15 (1999) of `setserial` this shell−script is not edited but instead gets its data from a configuration file: `/etc/serial.conf`. Furthermore you may not even need to edit serial.conf because using the "setserial" command on the command line may automatically cause serial.conf to be edited appropriately.

This was intended to make it so that you don't need to edit any file in order to set up (or change) setserial so it will do the right thing each time that Linux is booted. But there are serious pitfalls because it's not really "setserial" that edits serial.conf. Confusion is compounded because different distributions handle this differently. In addition, you may modify it so it works differently.

What often happens is this: When you shut down your PC the script that runs "setserial" at boot−time is run again, but this time it only does what the part for the "stop" case says to do: It uses "setserial" to find out what the current state of "setserial" is and puts that info into the `serial.conf` file. Thus when you run "setserial" to change the serial.conf file, it doesn't get changed immediately but only when and if you shut down normally.

Now you can perhaps guess what problems might occur. Suppose you don't shut down normally (someone turns the power off, etc.) and the changes don't get saved. Suppose you experiment with "setserial" and forget to run it a final time to restore the original state (or make a mistake in restoring the original state). Then your "experimental" settings are saved.

If you manually edit serial.conf, then your editing is destroyed when you shut down because it gets changed back to the state of setserial at shutdown. There is a way to disable the changing of serial.conf at shutdown and that is to remove "###AUTOSAVE###" or the like from first line of serial.conf. In at least one distribution, the removal of "###AUTOSAVE###" from the first line is automatically done after the first time you shutdown just after installation. The serial.conf file will hopefully contain some comments to help you out.

The file most commonly used to run setserial at boot−time (in conformance with the configuration file) is now /etc/init.d/setserial (Debian) or /etc/init.d/serial (Redhat), or etc., but it should not normally be edited. For 2.15 Redhat 6.0 just had a file /usr/doc/setserial−2.15/rc.serial which you have to move to /etc/init.d/ if you want setserial to run at boot−time.

To disable a port, use `setserial` to set it to "uart none". The format of /etc/serial.conf appears to be just like that of the parameters placed after "setserial" on the command line with one line for each port. If you don't use autosave, you may edit /etc/serial.conf manually.

BUG: As of July 1999 there is a bug/problem since with ###AUTOSAVE### only the setserial parameters displayed by "setserial −Gg /dev/ttyS*" get saved but the other parameters don't get saved. Use the −a flag to "setserial" to see all parameters. This will only affect a small minority of users since the defaults for the parameters not saved are usually OK for most situations. It's been reported as a bug and may be fixed by now.

In order to force the current settings set by setserial to be saved to the configuration file (serial.conf) without shutting down, do what normally happens when you shutdown: Run the shell−script `/etc/init.d/{set}serial stop`. The "stop" command will save the current configuration but the serial ports still keep working OK.

In some cases you may wind up with both the old and new configuration methods installed but hopefully only one of them runs at boot−time. Debian labeled obsolete files with "...pre−2.15".

IRQs

By default, both ttyS0 and ttyS2 will share IRQ 4, while ttyS1 and ttyS3 share IRQ 3. But actually sharing serial interrupts (using them in running programs) is not permitted unless you: 1. have kernel 2.2 or better, and 2. you've complied in support for this, and 3. your serial hardware supports it. See

Interrupt sharing and Kernels 2.2+ If you only have two serial ports, ttyS0 and ttyS1, you're still OK since IRQ sharing conflicts don't exist for non−existent devices.

If you add an internal modem and retain ttyS0 and ttyS1, then you should attempt to find an unused IRQ and set it both on your serial port (or modem card) and then use setserial to assign it to your device driver. If IRQ 5 is not being used for a sound card, this may be one you can use for a modem. To set the IRQ in hardware you may need to use isapnp, a PnP BIOS, or patch Linux to make it PnP. To help you determine which spare IRQ's you might have, type "man setserial" and search for say: "IRQ 11".

9.2 What is isapnp ?

`isapnp` is a program to configure Plug−and−Play (PnP) devices on the ISA bus including internal modems. It comes in a package called "isapnptools" and includes another program, "pnpdump" which finds all your ISA PnP devices and shows you options for configuring them in a format which may be added to the PnP configuration file: /etc/isapnp.conf. It may also be used with the −−dumpregs option to show the current IO address and IRQ of the modem's serial port. The isapnp command may be put into a startup file so that it runs each time you start the computer and thus will configure ISA PnP devices. It is able to do this even if your BIOS doesn't support PnP. See Plug−and−Play−HOWTO.

9.3 What is wvdialconf ?

`wvdialconf` will try to find which serial port (ttyS?) has a modem on it. It also creates a configuration program for the wvdial program. `wvdial` is used for simplified dialing out using the PPP protocol to an ISP. But you don't need to install PPP in order to use `wvdialconf`. It will only find modems which are not in use. It will also automatically devise a "suitable" init strings but sometimes gets it wrong. Since this command has no options, it's simple to use but you must give it the name of a file to put the init string (and other data) into. For example type: wvdialconf my_config_file_name.

9.4 What is stty ?

`stty` is like setserial but it sets the baud rate and other parameters of a serial port. Typing "stty −a < /dev/ttyS2" should show you how ttyS2 is configured. Most of the settings are for things that you never need to use with modems (such as some used only for old terminals of the 1970s). Your communication package should automatically set up all the setting correctly for modems. But stty is sometimes useful for trouble−shooting.

Two items set by stty are: 1. Hardware flow control by "crtscts" and 2. Ignore the CD signal from the modem: "clocal". If the modem is not sending a CD signal and clocal is disabled (stty shows −clocal) then a program may not be able to open the serial port. If the port can't open, the program may just hang, waiting (often in vain) for a CD signal from the modem.

Minicom sets clocal automatically when it starts up so there is no problem. But version 6.0.192 of Kermit hung when I set −clocal and tried to "set line ..." If −clocal is set and there is no CD signal then even the "stty" command will hang and there is seemingly no way to set clocal (except by running minicom). But minicom will restore −clocal when it exits. One way to get out of this is to use minicom to send the "AT&C" to the modem (to get the CD signal) and then exit minicom with no reset so that the CD signal remains on. Then you may use stty again.

---

10. Trying Out Your Modem (Dialing Out)

10.1 Are You Ready to Dial Out ?

Once you've plugged in your modem and know which serial port it's on you're ready to try using it. Before you try to get the Internet on it or have people call in to you, you could first try something simpler like dialing out to some number to see if your modem is working OK. Find a phone number that is connected to a modem. If you don't know what number to call, find out if a local library has a phone number for an on−line catalog.

Then make sure you are ready to phone. Do you know what serial port (such as ttyS2) your modem is on? You should have found this out when you io−irq configured your serial ports. Have you decided what speed you are going to use for this port? See Speed Table for a quick selection or What Speed Should I Use for more details. If you have no idea what speed to set, just set it a few times faster than the advertised speed of your modem. Also remember that if you see a menu where an option is "hardware flow control" and/or "RTS/CTS" or the like, select it. Is a live telephone cable plugged in to your modem? You may want to connect the cable to a real telephone to make sure that it can produce a dial tone.

Now you need to select a communication (dialing) program to use to dial out. Dialing programs include: minicom, seyon (X−windows), and kermit. See section Communications Programs about some communications programs. Two examples are presented next: Dialing Out with Minicom and Dialing Out with Kermit

10.2 Dialing Out with Minicom

Minicom comes with most Linux distributions. To configure it you should be the root user. Type "minicom −s" to configure. This will take you directly to the configuration (set−up) menus. Alternatively you could just run "minicom" and then type ^A to see the bottom status line. This shows to type ^A Z for help (you've already typed the ^A so just type z). From the help menu go to the Configuration menu.

Most of the options don't need to be set for just simply dialing out. To configure you have to supply a few basic items: the name of the serial port your modem is on such as /dev/ttyS2 and the speed such as 115200. These are set at the serial port menu. Go to it and set them. Also (if possible) set hardware flow control (RTS/CTS). Then save them. When typing in the speed, you should also see something like "8N1" which you should leave alone. It means: 8−bit bytes, No parity, 1 stop−bit appended to each byte. If you can't find the speed you want, a lower speed will always work for a test. Exit (hit return) when done and save the configuration as default (dfl) using the menu. You may want to exit minicom and start it again so it can now find the serial port and initialize the modem, or you could go to help and tell minicom to initialize the modem.

Now you are ready to dial. But first at the main screen you get after you first type "minicom" make sure there's a modem there by typing AT and then hit the <enter> key. It should display OK. If it doesn't something is wrong and there is no point of trying to dial.

If you got the "OK" go back to help and select the dialing directory. You may edit it and type in a phone number, etc. into the directory and then select "dial" to dial it. Alternatively, you may just dial manually (by selecting "manual" and then type the number at the keyboard). If it doesn't work, carefully note any error messages and try to figure out what went wrong.

10.3 Dialing Out with Kermit

You can find the latest version of `kermit` at `http://www.columbia.edu/kermit/`. For example, say your modem was on `ttyS3`, and its speed was 115200 bps. You would do the following:

```
linux# kermit
C-Kermit 6.0.192, 6 Sep 96, for Linux
 Copyright (C) 1985, 1996,
  Trustees of Columbia University in the City of New York.
Default file-transfer mode is BINARY
Type ? or HELP for help.
C-Kermit>set line /dev/ttyS3
C-Kermit>set carrier-watch off
C-Kermit>set speed 115200
/dev/ttyS3, 115200 bps
C-Kermit>c
Connecting to /dev/ttyS3, speed 115200.
The escape character is Ctrl-\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options.
ATE1Q0V1                            ; you type this and then the Enter key
OK                                  ; modem should respond with this
```

If your modem responds to AT commands, you can assume your modem is working correctly on the Linux side. Now try calling another modem by typing:

```
ATDT7654321
```

where 7654321 is a phone number. Use ATDP instead of ATDT if you have a pulse line. If the call goes through, your modem is working.

To get back to the kermit prompt, hold down the Ctrl key, press the backslash key, then let go of the Ctrl key, then press the C key:

```
Ctrl−\−C
(Back at linux)
C−Kermit>quit
linux#
```

This was just a test using the primitive "by−hand" dialing method. The normal method is to let kermit do the dialing for you with its built−in modem database and automatic dialing features, for example using a US Robotics (USR) modem:

```
linux# kermit
C−Kermit 6.0.192, 6 Sep 1997, for Linux
 Copyright (C) 1985, 1996,
  Trustees of Columbia University in the City of New York.
Default file−transfer mode is BINARY
Type ? or HELP for help
C−Kermit>set modem type usr        ; Select modem type
C−Kermit>set line /dev/ttyS3       ; Select communication device
C−Kermit>set speed 115200          ; Set the dialing speed
C−Kermit>dial 7654321              ; Dial
 Number: 7654321
 Device=/dev/ttyS3, modem=usr, speed=115200
 Call completed.<BEEP>
Connecting to /dev/ttyS3, speed 115200
The escape character is Ctrl−\ (ASCII 28, FS).
Type the escape character followed by C to get back,
or followed by ? to see other options.

Welcome to ...

login:
```

11. Dial−In

11.1 Dial−In Overview

Dial−in is where you set up your PC so that others may dial in to your PC (at your phone number) and use your PC. Unfortunately some use the term "dial−in" when what they actually mean is just the opposite: dial−out. Dial−in works like this. Someone with a modem dials your telephone number. Your modem answers the phone ring and connects. Once the caller is connected the getty program is notified and starts the login process for the caller. After the caller has logged in, the caller then may use your PC. It could be almost as if they were sitting at the monitor−console.

The caller may use a script to automatically log in. This script will be of the expect−send type. For example it expects "login:" and then (after it detects "login:") will send the users login name. It next expects the password and then sends the password, etc. Then once the user has been automatically logged in, the /etc/passwd (password file) might specify that a shell (such as bash) will be started for the user. Or it might specify that PPP is to start so that the user may be connected to the Internet. See the PPP−HOWTO for more details. The program that you use at your PC to handle dialin is called `getty` or `mgetty`. See Getty

An advanced getty program such as mgetty can watch to see if PPP is started by the PC on the other end. If so, the login prompt would be skipped, a PPP connection would be made, and login would take place automatically over the PPP connection.

11.2 What Happens when Someone Dials In ?

Here's a more detailed description of dialin. This all assumes that you are using either mgetty or uugetty. Agetty is inferior and doesn't work exactly as described below (see About agetty)

For dialin to work, the modem must be listening for a ring and getty must be running and ready to respond to the call. Your modem is normally listening for incoming calls, but what it does when it gets a ring depends on how it's configured. The modem can either automatically answer the phone or not directly answer it. In the latter case the modem sends a "RING" message to getty and then getty tells the modem to answer the ring. In either case, it may be set up to answer on say the 4th ring, which allows one to walk/run to the phone and pick it up manually before the 4th ring. One would do this if they expect that the call is a voice call from a real person (and not from another modem).

Once the call is answered, your modem sends tones to the other modem (and conversely). The two modems negotiate how they will communicate and when this is completed your modem sends a "CONNECT" message (or the like) to `getty`. When `getty` gets this message, it sends a login prompt out the serial port. Once a user name is given to this prompt `getty` may just call on a program named `login` to handle the logging in from there on. `getty` usually starts running at boot−time but it should wait until a connection is made before sending out a "login" prompt.

Now for more details on the two methods of answering the call. For the first method where the modem automatically answers the call, the number of times it will ring before answering is controlled by the S0 register of the modem. If S0 is set to 3, the modem will automatically answer on the 3rd ring. If it's set to 0 then the modem will only answer the call if getty sends it an "A" (= Answer) AT command to the modem while the phone is ringing. (Actually an "ATA" is sent since all modem commands are prefixed by "AT".) This is known as "manual" answering since the modem itself doesn't do it automatically (but getty does). Since it's not really manual, "manual" appears in quotes. You might think it best to utilize the ability of the modem to automatically answer the call, but it's actually better if `getty` answers it "manually".

For the "manual" answer case, `getty` opens the port at boot−time and listens. When the phone rings, a "RING" message is sent to the listening `getty`. Then if `getty` wants to answer this ring, it sends the modem an "A" command. Note that getty may be set to answer only after say 4 "RING" messages (the 4th ring) similar to the automatic answer method. The modem then makes a connection and sends a "CONNECT ..." message to `getty` which then sends a login prompt to the caller.

The automatic answer case uses the CD (Carrier Detect aka DCD) wire from the modem to the serial port to tell when a connection is made. It works like this. At boot−time `getty` tries to open the serial port but the attempt fails since the modem has negated CD (the modem is idle). Then the `getty` program waits at the open statement in the program until a CD signal is asserted. When a CD signal arrives (perhaps hours later) then the port is opened and `getty` sends the login prompt. While `getty` is waiting (sleeping) at the open statement, other processes can run so it doesn't degrade computer perfomance. What actually wakes `getty` up is an interrupt which is issued when the CD line from the modem changes its state to on.

You may wonder how getty is able to open the serial port in the "manual"−answer case since CD may be negated. Well, there's a way to write a program to force the port to open even if there is no CD signal asserted.

11.3 56k doesn't work

If you expect that people will be able to dial−in to you at 56k, it can't be done unless you have all the following:

1. You have a digital connection to the telephone company such as a trunkside−T1 or ISDN line
2. You use special digital modems (see Digital Modems)
3. You have a "... concentrator", or the like to interface your digital−modems to the digital lines of the telephone company.

A "... concentrator" may be called a "modem concentrator" or a "remote access concentrator" or it could be included in a "remote access server" (RAS) which includes the digital modems, etc. This type of setup is used by ISPs (Internet Service Providers).

11.4 Getty

Introduction to Getty

`getty` is the program you run for dialin. You don't need it for dialout. In addition to presenting a login prompt, it also may help answer the telephone. Originally getty was used for logging in to a computer from a dumb terminal. A major use of it today is for logging in to a Linux console. There are several different getty programs but a few of these work OK with modems for dialin. The getty program is usually started at boot−time. It must be called from the /etc/inittab file. In this file you may find some examples which you will likely need to edit a bit. Hopefully these examples will be for the flavor of getty installed on your PC.

There are four different getty programs to choose from that may be used with modems for dial−in: `mgetty`, `uugetty`, `getty_em`, and `agetty`. A brief overview is given in the following subsections. `agetty` is the weakest of the four and it's mainly for use with directly connected text−terminals. `mgetty` has support for fax and voice mail but `Uugetty` doesn't. `mgetty` allegedly lacks a few of the features of `uugetty`. `getty_em` is a simplified version of `uugetty`. Thus `mgetty` is likely your best choice unless you are already familiar with `uugetty` (or find it difficult to get `mgetty`). The syntax for these getty programs differs, so be sure to check that you are using the correct syntax in `/etc/inittab` for whichever getty you use.

In order to see what documentation exists about the various gettys on your computer, use the "locate" command. Type: locate "*getty*" (including the quotes may help). Note that many distributions just call the program getty even though it may actually be agetty, uugetty, etc. But if you read the man page (type: man getty), it might disclose which getty it is. This should be the getty program with path `/sbin/getty.`

Getty "exits" after login (and can respawn)

After you log in you will notice (by using "top" or "ps −ax") that the getty process is no longer running. What happened to it? Why does getty restart again if your shell is killed? Here's why.

After you type in your user name, getty takes it and calls the login program telling it your user name. Then the getty process is replaced by the login process. The login process asks for your password, checks it and starts whatever process is specified in your password file. This process is often the bash shell. Bash starts and replaces the login process. Note that one process replaces another and that the bash shell process originally started as the getty process. The implications of this will be explained below.

Now in the /etc/inittab file getty is supposed to respawn (restart) if killed. It says so on the line that calls getty. But if the bash shell (or the login process) is killed, getty respawns (restarts). Why? Well, both the login process and bash are replacements for getty and inherit the signal connections establish by their predecessors. In fact if you observe the details you will notice that the replacement process will have the same process ID as the original process. Thus bash is sort of getty in disguise with the same process ID number. If bash is killed it is just like getty was killed (even though getty isn't running anymore). This results in getty respawning.

When one logs out, all the processes on that serial port are killed including the bash shell. This will also happen (when enabled) if a hangup signal is sent to the serial port by a drop of DCD voltage by the modem. Either the logout or drop in DCD will result in getty respawning. One may force getty to respawn by manually killing bash (or login) either by hitting the k key while in "top" or with the kill command. You will likely need to kill it with signal 9 (which can't be ignored).

About mgetty

`mgetty` was written as a replacement for `uugetty` which was in existence long before `mgetty`. Both are for use with modems. `mgetty` may be also used for directly connected terminals. In addition to allowing dialup logins, `mgetty` also provides FAX support and auto PPP detection. There is a supplemental program called `vgetty` which handles voicemail for some modems. `mgetty` documentation is fair (except for voice mail), and is not supplemented in this HOWTO. To automatically start PPP one must edit /etc/mgetty/login.conf to enable "AutoPPP" You can find the latest information on `mgetty` at http://www.leo.org/~doering/mgetty/ and http://alpha.greenie.net/mgetty

About uugetty

`getty_ps` contains two programs: `getty` is used for console and terminal devices, and `uugetty` for modems. Greg Hankins (former author of Serial−HOWTO) used `uugetty` so his writings about it are included here. See Uugetty.

About getty_em

This is a simplified version of ``uugetty''. It was written by Vern Hoxie after he became fully confused with complex support files needed for getty_ps and uugetty.

It is part of the collection of serial port utilities and information by Vern Hoxie available via ftp from scicom.alphacdc.com/pub/linux. The name of the collection is ``serial_suite.tgz''.

About agetty

This subsection is long since the author tried using agetty for dialin. `agetty` is seemingly simple since there are no initialization files. But when I tried it, it opened the serial port even when there was no CD signal present. It then sent both a login prompt and the /etc/issue file to the modem in the AT−command state before a connection was made. The modem thinks all this an AT command and if it does contain any "at" strings (by accident) it is likely to adversely modify your modem profile. Echo wars can start where getty and the modem send the same string back and forth over and over. You may see a "respawning too rapidly" error message if this happens. To prevent this you need to disable all echoing and result codes from the modem (E0 and Q1). Also use the −i option with agetty to prevent any /etc/issue file from being sent.

If you start getty on the modem port and a few seconds later find that you have the login process running on that port instead of getty, it means that a bogus user name has been sent to agetty from the modem. To keep this from happening, I had to save my dial−in profile in the modem so that it become effective at power−on. The other saved profile is for dial−out. Then any dial−out programs which use the modem must use a Z, Z0, or Z1 in their init string to initialize the modem for dial−out (by loading the saved dial−out profile). If the 1−profile is for dial−in you use Z1 to load it, etc. If you want to listen for dial−in later on, then the modem needs to be reset to the dial−in profile. Not all dial−out programs can do this reset upon exit from them.

Thus while agetty may work OK if you set up a dial−in profile correctly in the modem hardware, it's probably best suited for virtual consoles or terminals rather than modems. If agetty is running for dialin, there's no easy way to dial out. When someone first dials in to agetty, they should hit the return key to get the login prompt. `agetty` in the Debian distribution is just named `getty`.

About mingetty, and fbgetty

`mingetty` is a small getty that will work only for monitors (the usual console) so you can't use it with modems for dialin. `fbgetty` is as above but supports framebuffers.

11.5 Why "Manual" Answer is Best

The difference between the two ways of answering is exhibited when the computer happens to be down but the modem is still working. For the manual case, the "RING" message is sent to getty but since the computer is down, getty isn't there and the phone never gets answered. There are no telephone charges when there is no answer. For the automatic answer case, the modem (which is still on) answers the phone but no login message is ever sent since the computer is down. The phone bill runs up as the waiting continues. If the phone call is toll−free, it doesn't make much difference, although it may be frustrating waiting for a login prompt that never arrives. `mgetty` uses manual answer. `Uugetty` can do this too using a configuration script.

11.6 Ending a Dial−in Call

Caller logs out

When the call is over the normal way to end the connection is for the user to log out. This will kill the remote user's shell on your PC. As a result, getty will respawn and a hangup signal (negates DTR) will be sent from your serial port to the modem. This will only happen if stty −a shows hupcl (hang up on close) but this should be the default. The modem getting this hangup (negated DTR signal) will then hang up the phone line (provided the modem has been configured to do this −−see below). The modem should then be ready to answer any new incoming calls. Killing the user's shell causes getty to respawn and wait for the next call.

As an alternative to using DTR to tell the modem to hang up the phone line, a script used after getty respawns may send the unique escape code sequence +++ to the modem to put it into AT command mode. The +++ must have both an initial and final time delay. Once in AT command mode, a hangup command (H0) may be sent to the modem as well as other AT commands. If the PC fails to successfully signal the modem when a logout happens (or to use the +++ escape when restarting getty), then the modem is apt to remain in on−line mode and no more incoming calls can be received.

When DTR drops (is negated)

When DTR (the "hang−up" signal) is negated, what the modem does depends on the value of the &D option in the modem's profile. If it's &D0 nothing at all happens (the modem ignores the negation of DTR).

`&D2`**:** The modem will hang up and go into AT command mode (off−line) to wait for the next call. Except that it will not automatically answer the phone (if it should) until DTR is asserted again. But since getty is set to respawn (in /etc/inittab) then getty will immediately restart after a logout and this will assert DTR. So what happens when someone logs out is that DTR only is negated for a fraction of a second (winks) before it gets asserted again. The DTR must be negated for at least the time specified by register S25.

`&D3`**:** In this case the modem does a hard reset: It hangs up and restores the saved profile as specified by &Y. It should now be in the same state it was in when first powered on and it's ready for incoming calls. The S25 limit may have no effect so even a very short "wink" is detected. Another brand of modem says the S25 limit is still valid. Thus `&D3` is a stronger "reset" than `&D2` which doesn't restore the saved profile and could require a longer wink to work.

If you don't know which of the above two to use try using `&D3` first. Under favorable conditions, either one should work OK. It's reported that some modems require `&D2`.

Caller hangs up

Instead of logging out the normal way, a caller may simply hang up. This results in a lost connection and of course a loss of carrier. Other problems could also cause a loss of carrier. The "NO CARRIER" result code is displayed. The modem hangs up and waits for the next call. Except that there is no getty running yet to start the login process.

Here's how getty gets started again: The loss of carrier should negate the CD signal sent by the modem to the serial port (provided `&C1` has been set). When the the PC's serial port gets the negated CD signal it should kill the shell and then getty should respawn.

This paragraph is about other things that happen but do nothing. Only the curious need read it. When the shell is killed a DTR wink is sent to the modem but since the modem is not on−line anymore and has already hung up, the modem ignores the negation of DTR (hang up). The loss of carrier also negates the DSR signal sent by the modem to the serial port (provided &S1 or &S2 is set) but this signal is ignored (by Linux).

11.7 Dial−in Modem Configuration

The getty programs have a provision for sending an init string to the modem to configure it. But you may need to edit it. Another method is to save a suitable init string inside the modem (see Init Strings: Saving and Recalling for how to save it in the modem).

The configuration for dial−in depends both on the getty you use and perhaps on your modem. If you can't find suggested configurations in other documentation here are some hints using Hayes AT commands:

- `&C1` Make the CD line to the serial port track the actual state of the carrier (CD asserted only when there's carrier). Getty_em requires &C0 (CD always asserted)
- `&D3` Do a hard reset of the modem when someone logs out (or hangs up). For some modems it's reported that &D2 is required since they can't tolerate a hard reset ??
- `E0` Don't echo AT commands back to the serial port. This is a must for agetty. Some suggest E1 (echo AT commands) for mgetty. For dial−out you want E1 so you can see what was sent.
- `&K3` Use hardware flow control
- `Q0` Echo results words (such as CONNECT). Most gettys use them. But it's reported an AT&T version of uugetty and agetty require Q2 (no result words for dial−in).
- `S0=?` mgetty suggests S0=0 (manual answer). If you set S0=3 the modem will auto−answer on the 3rd ring, etc. Agetty uses auto−answer. So does uugetty (usually).
- `V1` Display results (such as CONNECT) in words (and not in code)
- `X4` Check for dialtone and busy signal

11.8 Callback

Callback is where someone first dials in to your modem. Then, you get a little info from the caller and then call it right back. Why would you want to do this? One reason is to save on telephone bills if you can call the caller cheaper than the caller can call you. Another is to make sure that the caller really is who it claims to be. If a caller calls you and claims to be calling from its usual phone number, then one way to verify this is to actually place a new call to that number.

There's a program for Linux called "callback" that works with mgetty. It's at ftp://ftp.icce.rug.nl/pub/unix/. Step−by−step instructions on how someone installed it (and PPP) is at http://www.stokely.com/unix.serial.port.resources/callback.html

11.9 Voice Mail

Voice mail is like an answering machine run by a computer. To do this you must have a modem that supports "voice" and supporting software. Instead of storing the messages on tape, they are stored in digital format on a disk. When a person phones you, they hear a "greeting" message and can then leave a message for you. More advanced systems would have caller−selectable mail boxes and caller−selectable messages to listen to. Free software is available in Linux for simple answering, but doesn't seem to be available yet for the more advanced stuff.

I know of two different voicemail packages for Linux. One is a very minimal package (see Voicemail Software). The other, more advanced, but currently poorly documented, is `vgetty`. It's an optional addition to the well documented and widely distributed `mgetty` program. It supports ZyXEL−like voice modem commands. In the Debian distribution, you must get the mgetty−voice package in addition to the mgetty package and mgetty−doc package. Obsolete documentation has been removed from mgetty but replacement documentation is lacking (except if you use the −h (help) option when running certain programs, etc.). But one sees postings about using it on the mgetty newsgroup. See About mgetty. It seems that `vgetty` is currently not very stable but it's successfully being used and development of it continues. If this is the latest version of this HOWTO can someone who is familiar with vgetty please let me know its current status.

11.10 Simple Manual Dial−In

This is really doing it manually! There's a way to answer a call without bothering to edit any configuration files for dial−in or enabling getty but the caller can't login. To do this you run a terminal program such as minicom. Make sure it's connected to your modem by typing "AT <enter>" and expect "OK". Then wait for the call. Then you really answer the call manually by typing "ATA" when the phone is ringing. This doesn't run getty and the caller can't login. But if the caller is calling in with a terminal program they may type a message to your screen (and conversely). You both may send files back and forth by using the commands built into the terminal programs (such as minicoms). Another way to answer such a call would be to type say "ATS0=3" just before the call comes in to enable the modem to auto−answer on the third ring.

This is one way to crudely transfer files with someone on a MS Windows PC who uses HyperTerminal or Terminal (for Windows 3.x or DOS). These two MS programs are something like minicom. Using this simple manual method (for Linux−to−Linux or MS−to−Linux) requires two people to be present, one one each end of the phone line connection running a terminal communications program. Be warned that if both people type at the same time it's chaos. It's a "last resort" way to transfer files between any two people that have PCs (either Linux or MS Windows). It could also be used for testing your modem or as a preliminary test before setting up dial−in.

11.11 Complex GUI Dial−In, VNC

At the opposite extreme to the simple (but labor intensive) manual dial−in is one that results in GUI graphical interface to the Linux PC. This generally requires that a network running TCP/IP protocol exist between the two computers. One way to get such a "network" is to dial−out to a PC set for dial−in and then run PPP on the phone line. PPP will use TCP/IP protocol encapsulated inside the PPP packets. Both sides must run PPP and mgetty can be configured to start PPP as soon as the caller does. The caller may use a PPP−dialer program just like they were dialing an ISP. Programs such as wvdial, eznet, or chat scripts should do it.

Instead of this tiny network over a phone connection a much larger network (the entire world) is reached via an ISP. For their lowest−rate service many of them use proxy servers that will not give you access to the ports you need to use. Even if they don't use proxy servers, the IP address they give you is only temporary for the session, so you'll need to email this IP to whomever wants to reach you. If you get a more expensive ISP service, then you can avoid these problems.

One way to get a GUI interface from the remote PC is to run the GPLed program: Virtual Network Computer (VNC) from AT&T. It has a server part which you run on your Linux PC for dial−in and a viewer (client) part used for dial−out. Neither of these actually does any dialing or login but assumes that you have a network already set up. The VNC server has an X−server built in and may use Linux's twm window manager. See the article on VNC in Linux Magazine: http://www.linux−mag.com/2000−11/desktop_03.html. The AT&T site for VNC is: http://www.uk.research.att.com/vnc/.

With VNC one can also connect to remote Windows PCs, get the Windows GUI on a Linux PC, and run Windows programs on the remote Windows PC. Of course the Windows PC must be running VNC (as a server). Obviously, a GUI connection over a modem will be slower than a text−only connection especially if you run KDE or GNOME or want 16−bit color.

11.12 Interoperability with MS Windows

Once you have dial−in set up, others may call in to you using minicom (or the like) from Unix−like systems. From MS Windows one may call you using "HyperTerminal (or just "Terminal" in Windows 3.1 or DOS).

If in Windows one wants to use dial−up with a network protocol over the phone line it's called "Dial−up Networking". But it probably will not be able to communicate with Linux. For setting up such dial−in in Windows one clicks on "server" while dial−out is the "client. Such dial−in is often called "remote control" meaning that the caller can use your PC, run programs on it, and thus control it remotely.

While it's easy to call in to a text−based Linux system from MS Windows, it's not so easy the other way around (partly because Windows is not text−based and would need to put the caller into DOS where files wouldn't be protected like they are in Linux.

However Windows "Dial−up Networking" can establish a dial−in provided the caller uses certain network protocols over the phone line: MS's or Novel's (two protocols not liked by Linux). So if someone with Windows enables their Dial−up networking server in Windows 98, you can't just dial in directly to it from Linux. This type of dial−in doesn't permit the caller to run most of the programs on the host like Linux does. It's called "remote access" and one may transfer files, use the hosts printer, access databases, etc. Is there some way to interface to Dial−up Networking from Linux??

It is possible for two people to crudely chat and send files using Minicom on the Linux end and HyperTerminal on the Windows end. It's all done manually by two live persons, one on each end of the phone connection. See Simple Manual Dial−In.

At the opposite extreme, one would like to run a dial−in so that the person calling would get a GUI interface. For that a network protocol is normally used. It's possible using PC Anywhere for Windows or VNC for both Linux and Windows. But PC Anywhere doesn't seem to talk to Linux ?? Other Window programs for "remote control" include Laplink, Co−Session, and Microcom. Do any such programs support Linux besides VNC ??

---

12. Uugetty for Dial−In (from the old Serial−HOWTO)

Be aware that you could use mgetty as a (better?) alternative to `uugetty`. `mgetty` is newer and more popular than uugetty. See Getty for a brief comparison of these 2 gettys.

12.1 Installing getty_ps

Since uugetty is part of getty_ps you'll first have to install getty_ps. If you don't have it, get the latest version from `metalab.unc.edu:/pub/Linux/system/serial`. In particular, if you want to use high speeds (57600 and 115200 bps), you must get version 2.0.7j or later. You must also have libc 5.x or greater.

By default, `getty_ps` will be configured to be Linux FSSTND (File System Standard) compliant, which means that the binaries will be in `/sbin`, and the config files will be named `/etc/conf.{uu}getty.ttyS`*N*. This is not apparent from the documentation! It will also expect lock files to go in `/var/lock`. Make sure you have the `/var/lock` directory.

If you don't want FSSTND compliance, binaries will go in `/etc`, config files will go in `/etc/default/{uu}getty.ttyS`*N*, and lock files will go in `/usr/spool/uucp`. I recommend doing things this way if you are using UUCP, because UUCP will have problems if you move the lock files to where it isn't looking for them.

`getty_ps` can also use `syslogd` to log messages. See the man pages for `syslogd(1)` and `syslog.conf(5)` for setting up `syslogd`, if you don't have it running already. Messages are logged with priority LOG_AUTH, errors use LOG_ERR, and debugging uses LOG_DEBUG. If you don't want to use `syslogd` you can edit `tune.h` in the `getty_ps` source files to use a log file for messages instead, namely `/var/adm/getty.log` by default.

Decide on if you want FSSTND compliance and syslog capability. You can also choose a combination of the two. Edit the `Makefile`, `tune.h` and `config.h` to reflect your decisions. Then compile and install according to the instructions included with the package.

12.2 Setting up uugetty

With `uugetty` you may dial out with your modem while `uugetty` is watching the port for logins. `uugetty` does important lock file checking. Update `/etc/gettydefs` to include an entry for your modem. For help with the meaning of the entries that you put into `/etc/gettydefs`, see the "serial_suite" collected by Vern Hoxie. How to get it is in section See About getty_em. When you are done editing `/etc/gettydefs`, you can verify that the syntax is correct by doing:

```
linux# getty −c /etc/gettydefs
```

Modern Modems

If you have a 9600 bps or faster modem with data compression, you can lock your serial port to one speed. For example:

```
# 115200 fixed speed
F115200# B115200 CS8 # B115200 SANE −ISTRIP HUPCL #@S @L @B login: #F115200
```

If you have your modem set up to do RTS/CTS hardware flow control, you can add `CRTSCTS` to the entries:

```
# 115200 fixed speed with hardware flow control
F115200# B115200 CS8 CRTSCTS # B115200 SANE −ISTRIP HUPCL CRTSCTS #@S @L @B login: #F115200
```

Old slow modems

If you have a slow modem (under 9600 bps) Then, instead of one line for a single speed, your need several lines to try a number of speeds. Note the these lines are linked to each other by the last "word" in the line such as #4800. Blank lines are needed between each entry. Are the higher modem−to−serial_port speeds in this example really needed for a slow modem ?? The uugetty documentation shows them so I'm not yet deleting them.

```
# Modem entries
115200# B115200 CS8 # B115200 SANE −ISTRIP HUPCL #@S @L @B login: #57600

57600# B57600 CS8 # B57600 SANE −ISTRIP HUPCL #@S @L @B login: #38400

38400# B38400 CS8 # B38400 SANE −ISTRIP HUPCL #@S @L @B login: #19200

19200# B19200 CS8 # B19200 SANE −ISTRIP HUPCL #@S @L @B login: #9600
```

```
9600# B9600 CS8 # B9600 SANE -ISTRIP HUPCL #@S @L @B login: #4800

4800# B4800 CS8 # B4800 SANE -ISTRIP HUPCL #@S @L @B login: #2400

2400# B2400 CS8 # B2400 SANE -ISTRIP HUPCL #@S @L @B login: #1200

1200# B1200 CS8 # B1200 SANE -ISTRIP HUPCL #@S @L @B login: #115200
```

Login Banner

If you want, you can make `uugetty` print interesting things in the login banner. In Greg's examples, he has the system name, the serial line, and the current bps rate. You can add other things:

```
@B    The current (evaluated at the time the @B is seen) bps rate.
@D    The current date, in MM/DD/YY.
@L    The serial line to which uugetty is attached.
@S    The system name.
@T    The current time, in HH:MM:SS (24-hour).
@U    The number of currently signed-on users.  This is  a
      count of the number of entries in the /etc/utmp file
      that have a non-null ut_name field.
@V    The value of VERSION, as given in the defaults file.
To display a single '@' character, use either '\@' or '@@'.
```

12.3 Customizing uugetty

There are lots of parameters you can tweak for each port you have. These are implemented in separate config files for each port. The file `/etc/conf.uugetty` will be used by *all* instances of `uugetty`, and `/etc/conf.uugetty.ttySN` will only be used by that one port. Sample default config files can be found with the `getty_ps` source files, which come with most Linux distributions. Due to space concerns, they are not listed here. Note that if you are using older versions of `uugetty` (older than 2.0.7e), or aren't using FSSTND, then the default file will be `/etc/default/uugetty.ttySN`. Greg's `/etc/conf.uugetty.ttyS3` looked like this:

```
# sample uugetty configuration file for a Hayes compatible modem to allow
# incoming modem connections
#
# line to initialize
INITLINE=ttyS3
# timeout to disconnect if idle...
TIMEOUT=60
# modem initialization string...
# format: <expect> <send> ... (chat sequence)
INIT="" AT\r OK\r\n
WAITFOR=RING
CONNECT="" ATA\r CONNECT\s\A
# this line sets the time to delay before sending the login banner
DELAY=1
#DEBUG=010
```

Add the following line to your `/etc/inittab`, so that `uugetty` is run on your serial port, substituting in the correct information for your environment − run−levels (2345 or 345, etc.) config file location, port, speed, and default terminal type:

```
S3:2345:respawn:/sbin/uugetty -d /etc/default/uugetty.ttyS3 ttyS3 F115200 vt100
```

Restart `init`:

```
linux# init q
```

For the speed parameter in your `/etc/inittab`, you want to use the highest bps rate that your modem supports.

Now Linux will be watching your serial port for connections. Dial in from another machine and login to you Linux system.

`uugetty` has a lot more options, see the man page for `uugetty)` (often just called `getty`) for a full description. Among other things there is a scheduling feature, and a ringback feature.

---

13. What Speed Should I Use with My Modem?

By "speed" we really mean the "data flow rate" but almost everybody incorrectly calls it speed. For all modern modems you have no choice of the speed that the modem uses on the telephone line since it will automatically choose the highest possible speed that is feasible under the circumstances. If one modem is slower than the other, then the faster modem will operate at the slower ones speed. On a noisy line, this speed will drop still lower.

While the above speeds are selected automatically by the modems you do have a choice as to what speed will be used between your modem and your computer (PC−to−modem speed). This is sometimes called "DTE speed" where "DTE" stands for Data Terminal Equipment (Your computer is a DTE.) You need to set this speed high enough so this part of the signal path will not be a bottleneck. The setting for the DTE speed is the maximum speed of this link. Most of the time it should operate at lower speeds.

For an external modem, DTE speed is the speed (in bits/sec) of the flow over the cable between you modem and PC. For an internal modem, it's the same idea since the modem also emulates a serial port. It may seem ridiculous having a speed limit on communication between a computer and a modem card that is directly connected inside the computer to a much higher speed bus. But it's that way since the modem card probably includes a dedicated serial port which does have speed limits (and settable speeds).

13.1 Speed and Data Compression

What speed do you choose? If it were not for "data compression" one might try to choose a DTE speed exactly the same as the modem speed. Data compression takes the bytes sent to the modem from your computer and encodes them into a fewer number of bytes. For example, if the flow (speed) from the PC to the modem was 20,000 bytes/sec (bps) and the compression ratio was 2 to 1, then only 10,000 bytes/sec would flow over the telephone line. Thus for a 2:1 compression ratio you need to set the DTE speed to double the maximum modem speed on the phone line. If the compression ratio were 3 to 1 you need to set it 3 times faster, etc.

13.2 Where do I Set Speed ?

This DTE (PC−to−modem) speed is normally set by a menu in your communications program or by an option given to the getty command if someone is dialing in. You can't set the DCE modem−to−modem speed.

13.3 Can't Set a High Enough Speed

You need to find out the highest speed supported by your hardware. As of late 1998 (and late 2000) most serial ports only supported speeds up to 115.2k bps. Some 56k internal modems support 230.4k bps (but it may be hard to find out which ones do). Recent Linux kernels support high speeds (over 115.2k) but you might have some problems using it because of one or both of the following reasons:

1. Older versions of an application program (or stty) may not accept the high speed.
2. Setserial may show the wrong speed unless you use the baud_base option. In any case the high speed will work OK.

How speed is set in hardware: the divisor and baud_base

Here's a list of commonly used divisors and their corresponding speeds (assuming a maximum speed of 115,200): 1 (115.2k), 2 (57.6k), 3 (38.4k), 6 (19.2k), 12 (9.6k), 24 (4.8k), 48 (2.4k), 96 (1.2k), etc. The serial driver sets the speed in the hardware by sending the hardware only a "divisor" (a positive integer). This "divisor" divides the maximum speed of the hardware resulting in a slower speed (except a divisor of 1 obviously tells the hardware to run at maximum speed).

Normally, if you specify a speed of 115.2k (in your communication program or by stty) then the serial driver sets the port hardware to divisor 1 which obviously sets the highest speed. If you happen to have hardware with a maximum speed of say 230.4k, then specifying 115.2k will result in divisor 1 and will actually give you 230.4k. This is double the speed that you set. In fact, for any speed you set, the actual speed will be double. If you had hardware that could run at 460.8k then the actual speed would be quadruple what you set.

Work−arounds for setting speed

To correct this accounting (but not always fix the problem) you may use "setserial" to change the baud_base to the actual maximal speed of your port such as 230.4k. Then if you set the speed (by your application or by stty) to 230.4k, a divisor of 1 will be used and you'll get the same speed as you set. PROBLEM: stty and many communication programs (as of mid 1999) still have 115.2k as their maximum speed setting and will not let you set 230.4k, etc. So in these cases one solution is not to change anything with `setserial` but mentally keep in mind that the actual speed is always double what you set.

There's another work−around which is not much better. To use it you set the baud_base (with setserial) to the maximal speed of your hardware. This corrects the accounting so that if you set say 115.2k you actually get 115.2k. Now you still have to figure out how to set the highest speed if your communication program (or the like) will not let you do it. Fortunately, setserial has a way to do this: use the "spd_cust" parameter with "divisor 1". Then when you set the speed to 38400 in a communication program, the divisor will be set to 1 in the port and it will operate at maximum speed. For example:
setserial /dev/ttyS2 spd_cust baud_base 230400 divisor 1
Don't try using "divisor" for any other purpose other than the special use illustrated above (with spd_cust).

If there are two or more high speeds that you want to use that your communication program can't set, then it's not quite as easy as above. But the same principles apply. You could just keep the default baud_base and understand that when you set a speed you are really only setting a divisor. So your actual speed will always be your maximum speed divided by whatever divisor is set by the serial driver. See How speed is set in hardware: the divisor and baud_base

Crystal frequency is not baud_base

Note that the baud_base setting is usually much lower than the frequency of the crystal oscillator in the hardware since the crystal frequency is often divided by 16 in the hardware to get the actual top speed. The reason the crystal frequency needs to be higher is so that this high crystal speed can be used to take a number of samples of each bit to determine if it's a 1 or a 0.

13.4 Speed Table

It's best to have at least a 16650 UART for a 56k modem but few modems or serial ports provide it. Second best is a 16550 that has been tweaked to give 230,400 bps (230.4 kbps). Most people still use a 16550 that is only 115.2 kbps but it's claimed to only slow down thruput by a few percent (on average). This is because a typical compression ratio is 2 to 1 and for downloading compressed files (packages) it's 1 to 1. There's no degradation for these cases. Here are some suggested speeds to set your serial line if your modem speed is:

- 56k (v.90): use 115.2 kbps or 230.4 kbps (best)
- 33.6k (v.34bis): use 115.2 kbps
- 28.8k (v.34): use 115.2 kbps
- 14.4k (v.32bis): use 57600 bps
- 9.6k (v.32): use 38400 bps
- slower than a 9600 bps (v.32) modem: Set the speed to the same speed as the modem.

All the above speeds may use v.42bis data compression and v.42 error correction. If data compression is not used then the speed may be set lower so long as it's above the modem speed.

---

14. Communications Programs And Utilities

While PPP is used for Internet access you also need a dialer program that is designed to work with PPP. Such a dialer program will dial a phone number. When the other side answers the phone then three things happen: PPP is started at both ends and you get logged in (often automatically). The exact sequence of these 3 events may vary. Dialer programs for ppp include wvdial, chap scripts, kppp, and gnome−ppp.

There are also other dialer programs which can dial out directly (thru a modem) to local libraries, etc. This isn't the Internet. `minicom` is the most popular followed by `Seyon` (X−Windows only) and `Kermit`. People have likely also used these programs for dialing out with ppp for the Internet but it's not what they were originally designed for.

14.1 Minicom vs. Kermit

Minicom is only a communications program while Kermit is both a communications program and a file transfer protocol. But one may use the Kermit protocol from within Minicom (provided one has Kermit installed on one's PC). Minicom is menu based while Kermit is command line based (interactive at the special Kermit prompt). While the Kermit program is free software, the documentation is not all free. There is no detailed manual supplied and it is suggested that you purchase a book as the manual. However Kermit has interactive online help which tells all but lacks tutorial explanations for the beginner. Commands may be put in a script file so you don't have to type them over again each time. Kermit (as a communications program) is more powerful than Minicom.

Although all Minicom documentation is free, it's not as extensive as Kermit's. Since permission is required to include Kermit in a commercial distribution, and since the documentation is not entirely free, some distributions don't include Kermit. In my opinion it's easier to set up Minicom, there is less to learn, and you can still use kermit from within Minicom.

14.2 List of Communication Software

Here is a list of some communication software you can choose from, If they didn't come with your distribution they should be available via FTP. I would like comparative comments on the dialout programs. Are the least popular ones obsolete?

Least Popular Dialout

- `ecu` − a communications program
- `pcomm` − `procomm`−like communications program with zmodem
- `xc` − xcomm communication package

Most Popular Dialout

- PPP dialers for getting on the internet: `wvdial`, `eznet`, `chat`, `pon` (uses chat),
- `minicom` − `telix`−like communications program. Can work with scripts, zmodem, kermit
- C−Kermit − portable, scriptable, serial and TCP/IP communications including file transfer, character−set translation, and zmodem support
- `seyon` − X based communication program

Fax

By using a fax program, you may use most modems to send faxes. In this case you dial out directly and not via ppp and an ISP. You also pay any long−distance telephone charges. email is more efficient.

- `efax` a small fax program
- `hylafax` a large fax program based on the client−server model.
- `mgetty+fax` handles fax stuff and login for dial−ins

Voicemail Software

- mvm is a Minimal VoiceMail for Linux
- vgetty is an extension to mgetty that handles voicemail for some modems. It should come with recent releases of mgetty.

Dial−in (uses getty)

- `mgetty+fax` is for modems and is well documented (except for voicemail as of early 1999). It also handles fax stuff and provides an alternative to `uugetty`. It's incorporating voicemail (using vgetty) features. See About mgetty
- `uugetty` is also for modems. It comes as a part of the `ps_getty` package. See About getty_ps

Other

- `callback` is where you dial out to a remote modem and then that modem hangs up and calls you back (to save on phone bills).
- `SLiRP` and `term` provide a PPP−like service that you can run in user space on a remote computer with a shell account. See term and SLiRP for more details
- `ZyXEL` is a control program for ZyXEL U−1496 modems. It handles dialin, dialout, dial back security, FAXing, and voice mailbox functions.
- SLIP and PPP software can be found at `ftp://metalab.unc.edu/pub/Linux/system/network/serial`.
- Other things can be found on `ftp://metalab.unc.edu/pub/Linux/system/serial` and `ftp://metalab.unc.edu/pub/Linux/apps/serialcomm` or one of the many mirrors. These are the directories where serial programs are kept.

14.3 SLiRP and term

`SLiRP` and `term` are programs which are of use if you only have a dial−up shell account on a Unix−like machine and want to get the equivalent of a PPP account (or the like) without being authorized to have it (possibly because you don't want to pay extra for it, etc.). `SLiRP` is more popular than `term` which is almost obsolete.

To use `SLiRP` you install it in your shell account on the remote computer. Then you dial up the account and run SLiRP on the remote and PPP on your local PC. You now have a PPP connection over which you may run a web browser on your local PC such as Netscape, etc. There may be some problems as SLiRP is not as good as a real PPP account. Some accounts may provide SLiRP since it saves on IP addresses (You have no IP address while using SLiRP).

`term` is something like SLiRP only you need to run `term` on both the local and remote computer. There is no PPP on the phone line since `term` uses its own protocol. To use `term` from your PC you need to use a term−aware version of ftp to do ftp, etc. Thus it's easier to use SLiRP since the ordinary version of ftp works fine with SLiRP. There is an unmaintained Term HOWTO.

14.4 MS Windows

If you want someone who uses MS Windows to dial in to your Linux PC then if they use:

- Windows 3.x: use `Terminal`
- Windows 95/98/2000: use `HyperTerminal`

Third party dial−out programs include HyperTerminal Private Edition.

---

15. Troubleshooting

15.1 My Modem is Physically There but Can't be Found

The error messages could be something like "No modem detected", "Modem not responding", or (strange) "You are already online" (from Minicom). If you have installed an internal modem (serial port is builtin) or are using an external one and don't know what serial port it's connected to then the problem is to find the serial port. See My Serial Port is Physically There but Can't be Found. This section is about finding out which serial port has the modem on it.

There's a program that looks for modems on commonly used serial ports called "wvdialconf". Just type "wvdialconf <a−new−file−name>". It will create the new file as a configuration file but you don't need this file unless you are going to use "wvdial" for dialing. See What is wvdialconf ? Unfortunately, if your modem is in "online data" mode, wvdialconf will report "No modem detected" See No response to AT

Your problem could be due to a winmodem (or the like) which usually can't be used with Linux. See Software−based Modems (winmodems). The "setserial program may be used to detect serial ports but will not detect modems on them. Thus "wvdialconf" is best to try first.

Another way try to find out if there's a modem on a port is to start "minicom" on the port (after first setting up minicom for the correct serial port −−you will need to save the setup and then exit minicom and start it again). Then type "AT" and you should see OK (or 0 if it's set for "digit result codes"). The results may be:

- No response. See No response to AT
- It takes many seconds to get an expected truncated response (including only the cursor moving down one line). See Extremely Slow: Text appears on the screen slowly after long delays
- Some strange characters appear but they are not in response to AT. This likely means that your modem is still connected to something at the other end of the phone line which is sending some cryptic packets or the like.

No response to AT

The modem should send you "OK" in response to your "AT" which you type to the modem (using minicom or the like). If you don't see "OK" (and in most cases don't even see the "AT" you typed either) then the modem is not responding (often because what you type doesn't even get to the modem).

A common cause is that there is no modem on the serial port you are typing to. For the case of an internal modem, that serial port likely doesn't exist either. That's because the PnP modem card (which has a built−in serial port) has either not been configured (by isapnp or the like) or has been configured incorrectly. See My Serial Port is Physically There but Can't be Found.

If what you type is really getting thru to a modem, then the lack of response could be due to the modem being in "online data" mode where it can't accept any AT commands. You may have been using the modem and then abruptly disconnected (such as killing the process with signal 9). In that case your modem did not get reset to "command mode" where it can interact to AT commands. Thus the message from minicom "You are already online. Hangup first." Well, you are sort of online but you are may not be connected to anything over the phone line. Wvdial will report "modem not responding" for the same situation.

To fix this as a last resort you could reboot the computer. Another way to try to fix this is to send +++ to the modem to tell it to escape back to "command mode" from "online data mode". On both sides of the +++ sequence there must be about 1 second of delay (nothing sent during "guard time"). This may not work if another process is using the modem since the +++ sequence could wind up with other characters inserted in between them or after the +++ (during the guard time). Ironically, even if the modem line is idle, typing an unexpected +++ is likely to set off an exchange of control packets (that you never see) that will violate the required guard time so that the +++ doesn't do what you wanted. +++ is usually in the string that is named "hangup string" so if you command minicom (or the like) to hangup it might work. Another way to do this is to just exit minicom and then run minicom again.

15.2 "Modem is busy"

What this means depends on what program sent it. The modem could actually be in use (busy). Another cause reported for the SuSE distribution is that there may be two serial drivers present instead of one. One driver was built into the kernel and the second was a module.

In kppp, this message is sent when an attempt to get/set the serial port parameters fails. These parameters are the one you see if you give the "stty −a" command. It's similar to the "Input/output error" one may get when trying to use "stty −F /dev/ttySx". Although the port is already opened when you see this message, I think it's possible to open a non−existent device (or one with the wrong IRQ or IO address). Now getting certain port parameters (such as the speed) means communicating with the serial port hardware. So this error message may mean that there is no serial port there (or that the modem's serial port has an incorrect (or no) IRQ or I0 address. It could be a wrong ttySx number. If /dev/modem is used it should be linked to the correct ttySx. In these cases the error message should have said: "Modem can't be found" which really means that it's serial port (often built into the modem) can't be found.

15.3 I can't get near 56k on my 56k modem

There must be very low noise on the line for it to work at even close to 56k. Some phone lines are so bad that the speeds obtainable are much slower than 56k (like 28.8k or even slower). Sometimes extension phones connected to the same line can cause problems. To test this you might connect your modem directly at the point where the telephone line enters the building with the feeds for everything else on that line disconnected (if others can tolerate such a test).

15.4 Uploading (downloading) files is broken/slow

Flow control (both at your PC and/or modem−to−modem) may not be enabled. For the uploading case: If you have set a high DTE speed (like 115.2k) then flow from your modem to your PC may work OK but uploading flow in the other direction will not all get thru due to the telephone line bottleneck. This will result in many errors and the resending of packets. It may thus take far too long to send a file. In some cases, files don't make it thru at all.

For the downloading case: If you're downloading long uncompressed files or web pages (and your modem uses data compression) or if you've set a low DTE speed, then downloading may also be broken due to no flow control.

15.5 For Dial−in I Keep Getting "line NNN of inittab invalid"

Make sure you are using the correct syntax for your version of `init`. The different `init`'s that are out there use different syntax in the `/etc/inittab` file. Make sure you are using the correct syntax for your version of `getty`.

15.6 I Keep Getting: ``Id "S3" respawning too fast: disabled for 5 minutes"

Id "S3" is just an example. In this case look on the line which starts with "S3" in `/etc/inittab` and calls getty. This line causes the problem. Make sure the syntax for this line is correct and that the device (ttyS3) exists and can be found. If the modem has negated CD and getty opens the port, you'll get this error message since negated CD will kill getty. Then getty will respawn only to be killed again, etc. Thus it respawns over and over (too fast). It seems that if the cable to the modem is disconnected or you have the wrong serial port, it's just like CD is negated. All this can occur when your modem is chatting with getty. Make sure your modem is configured correctly. Look at AT commands `E` and `Q`.

If you use uugetty, verify that your `/etc/gettydefs` syntax is correct by doing the following:

```
linux# getty −c /etc/gettydefs
```

This can also happen when the `uugetty` initialization is failing. See section uugetty Still Doesn't Work.

15.7 My Modem is Hosed after Someone Hangs Up, or uugetty doesn't respawn

This can happen when your modem doesn't reset when DTR is dropped. Greg Hankins saw his RD and SD LEDs go crazy when this happened. You need to have your modem reset. Most Hayes compatible modems do this with `&D3`, but for USR Courier, SupraFax, and other modems, you must set `&D2` (and `S13=1` for USR Courier). Check your modem manual (if you have one).

15.8 uugetty Still Doesn't Work

There is a `DEBUG` option that comes with `getty_ps`. Edit your config file `/etc/conf.{uu}getty.ttyS`*N* and add `DEBUG=`*NNN*. Where *NNN* is one of the following combination of numbers according to what you are trying to debug:

```
D_OPT   001           option settings
D_DEF   002           defaults file processing
D_UTMP  004           utmp/wtmp processing
D_INIT  010           line initialization (INIT)
D_GTAB  020           gettytab file processing
D_RUN   040           other runtime diagnostics
D_RB    100           ringback debugging
D_LOCK  200           uugetty lockfile processing
D_SCH   400           schedule processing
D_ALL   777           everything
```

Setting `DEBUG=010` is a good place to start.

If you are running `syslogd`, debugging info will appear in your log files. If you aren't running `syslogd` info will appear in `/tmp/getty:ttyS`*N* for debugging `getty` and `/tmp/uugetty:ttyS`*N* for `uugetty`, and in `/var/adm/getty.log`. Look at the debugging info and see what is going on. Most likely, you will need to tune some of the parameters in your config file, and reconfigure your modem.

You could also try `mgetty`. Some people have better luck with it.

15.9 (The following subsections are in both the Serial and Modem HOWTOs)

15.10 My Serial Port is Physically There but Can't be Found

If a physical device (such as a modem) doesn't work at all it may mean that the device is not at the I/O address that setserial thinks it's at. It could also mean (for a PnP card) that is doesn't yet have an address. Thus it can't be found.

Check the BIOS menus and BIOS messages. For the PCI bus use lspci or scanpci. If it's an ISA bus PnP serial port, try "pnpdump −−dumpregs" and/or see Plug−and−Play−HOWTO. Using "scanport" will scan all ISA bus ports and may discover an unknown port that could be a serial port (but it doesn't probe the port). It could hang your PC. You may try probing with setserial. See Probing. If nothing seems to get thru the port it may be accessible but have a bad interrupt. See Extremely Slow: Text appears on the screen slowly after long delays. Use `setserial -g` to see what the serial driver thinks and check for IRQ and I0 address conflicts. Even if you see no conflicts the driver may have incorrect information (view it by "setserial" and conflicts may still exist.

If two ports have the same IO address then probing it will erroneously indicate only one port. Plug−and−play detection will find both ports so this should only be a problem if at least one port is not plug−and−play. All sorts of errors may be reported/observed for devices illegally "sharing" a port but the fact that there are two devices on the same a port doesn't seem to get detected (except hopefully by you). In the above case, if the IRQs are different then probing for IRQs with setserial might "detect" this situation by failing to detect any IRQ. See Probing.

15.11 Extremely Slow: Text appears on the screen slowly after long delays

It's likely mis−set/conflicting interrupts. Here are some of the symptoms which will happen the first time you try to use a modem, terminal, or serial printer. In some cases you type something but nothing appears on the screen until many seconds later. Only the last character typed may show up. It may be just an invisible <return> character so all you notice is that the cursor jumps down one line. In other cases where a lot of data should appear on the screen, only a batch of about 16 characters appear. Then there is a long wait of many seconds for the next batch of characters. You might also get "input overrun" error messages (or find them in logs).

For more details on the symptoms and why this happens see the Serial−HOWTO section: "Interrupt Problem Details".

If it involves Plug−and−Play devices, see also Plug−and−Play−HOWTO.

As a quick check to see if it really is an interrupt problem, set the IRQ to 0 with "setserial". This will tell the driver to use polling instead of interrupts. If this seems to fix the "slow" problem then you had an interrupt problem. You should still try to solve the problem since polling uses excessive computer resources.

Checking to find the interrupt conflict may not be easy since Linux supposedly doesn't permit any interrupt conflicts and will send you a /dev/ttyS?: Device or resource busy error message if it thinks you are attempting to create a conflict. But a real conflict can be created if "setserial" has told the kernel incorrect info. The kernel has been lied to and thus doesn't think there is any conflict. Thus using "setserial" will not reveal the conflict (nor will looking at /proc/interrupts which bases its info on "setserial"). You still need to know what "setserial" thinks so that you can pinpoint where it's wrong and change it when you determine what's really set in the hardware.

What you need to do is to check how the hardware is set by checking jumpers or using PnP software to check how the hardware is actually set. For PnP run either "pnpdump −−dumpregs" (if ISA bus) or run "lspci" (if PCI bus). Compare this to how Linux (e.g. "setserial") thinks the hardware is set.

15.12 Somewhat Slow: I expected it to be a few times faster

One reason may be that whatever is on the serial port (such as a modem, terminal, printer) doesn't work as fast as you thought it did. A 56k Modem seldom works at 56k and the Internet often has congestion and bottlenecks that slow things down. If the modem on the other end does not have a digital connection to the phone line (and uses a special "digital modem" not sold in most computer stores), then speeds above 33.6k are not possible.

Another possible reason is that you have an obsolete serial port: UART 8250, 16450 or early 16550 (or the serial driver thinks you do). See "What are UARTS" in the Serial−HOWTO.

Use "setserial −g /dev/ttyS*". If it shows anything less than a 16550A, this may be your problem. If you think that "setserial" has it wrong check it out. See What is Setserial for more info. If you really do have an obsolete serial port, lying about it to setserial will only make things worse.

15.13 The Startup Screen Show Wrong IRQs for the Serial Ports.

Linux does not do any IRQ detection on startup. When the serial module loads it only does serial device detection. Thus, disregard what it says about the IRQ, because it's just assuming the standard IRQs. This is done, because IRQ detection is unreliable, and can be fooled. But if and when setserial runs from a start−up script, it changes the IRQ's and displays the new (and hopefully correct) state on on the startup screen. If the wrong IRQ is not corrected by a later display on the screen, then you've got a problem.

So, even though I have my `ttyS2` set at IRQ 5, I still see

```
ttyS02 at 0x03e8 (irq = 4) is a 16550A
```

at first when Linux boots. (Older kernels may show "ttyS02" as "tty02" which is the same as ttyS2). You may need to use `setserial` to tell Linux the IRQ you are using.

15.14 "Cannot open /dev/ttyS?: Permission denied"

Check the file permissions on this port with "ls −l /dev/ttyS?"_ If you own the ttyS? then you need read and write permissions: crw with the c (Character device) in col. 1. It you don't own it then it should show rw− in cols. 8 & 9 which means that everyone has read and write permission on it. Use "chmod" to change permissions. There are more complicated ways to get access like belonging to a "group" that has group permission.

15.15 "Operation not supported by device" for ttyS?

This means that an operation requested by setserial, stty, etc. couldn't be done because the kernel doesn't support doing it. Formerly this was often due to the "serial" module not being loaded. But with the advent of PnP, it may likely mean that there is no modem (or other serial device) at the address where the driver (and setserial) thinks it is. If there is no modem there, commands (for operations) sent to that address obviously don't get done. See What is set in my serial port hardware?

If the "serial" module wasn't loaded but "lsmod" shows you it's now loaded it might be the case that it's loaded now but wasn't loaded when you got the error message. In many cases the module will automatically loaded when needed (if it can be found). To force loading of the "serial" module it may be listed in the file: /etc/modules.conf or /etc/modules. The actual module should reside in: /lib/modules/.../misc/serial.o.

15.16 "Cannot create lockfile. Sorry"

When a port is "opened" by a program a lockfile is created in /var/lock/. Wrong permissions for the lock directory will not allow a lockfile to be created there. Use "ls −ld /var/lock" to see if the permissions are OK: usually rwx for everyone (repeated 3 times). If it's wrong, use "chmod" to fix it. Of course, if there is no "lock" directory no lockfile can be created there. For more info on lockfiles see the Serial−HOWTO subsection: "What Are Lock Files".

15.17 "Device /dev/ttyS? is locked."

This means that someone else (or some other process) is supposedly using the serial port. There are various ways to try to find out what process is "using" it. One way is to look at the contents of the lockfile (/var/lock/LCK...). It should be the process id. If the process id is say 100 type "ps 100" to find out what it is. Then if the process is no longer needed, it may be gracefully killed by "kill 100". If it refuses to be killed use "kill −9 100" to force it to be killed, but then the lockfile will not be removed and you'll need to delete it manually. Of course if there is no such process as 100 then you may just remove the lockfile but in most cases the lockfile should have been automatically removed if it contained a stale process id (such as 100).

15.18 "/dev/tty? Device or resource busy"

This means that the device you are trying to access (or use) is supposedly busy (in use) or that a resource it needs (such as an IRQ) is supposedly being used by another device (the resource is "busy"). This message is easy to understand if it only means that the device is busy (in use). But it often means that a resource is in use. What makes it even more confusing is that in some cases neither the device not the resources that it needs are actually "busy".

The ``resource busy" part often means (example for `ttyS2`) ``You can't use `ttyS2` since another device is using ttyS2's interrupt." The potential interrupt conflict is inferred from what "setserial" thinks. A more accurate error message would be ``Can't use `ttyS2` since the setserial data (and kernel data) indicates that another device is using `ttyS2`'s interrupt". If two devices use the same IRQ and you start up only one of the devices, everything is OK because there is no conflict yet. But when you next try to start the second device (without quitting the first device) you get a "... busy" error message. This is because the kernel only keeps track of what IRQs are actually in use and actual conflicts don't happen unless the devices are in use (open). The situation for I/O address (such as 0x3f8) conflict is similar.

This error is sometimes due to having two serial drivers: one a module and the other compiled into the kernel. Both drivers try to grab the same resources and one driver finds them "busy".

There are two possible cases when you see this message:

1. There may be a real resource conflict that is being avoided.
2. Setserial has it wrong and the only reason `ttyS2` can't be used is that setserial erroneously predicts a conflict.

What you need to do is to find the interrupt setserial thinks `ttyS2` is using. Look at /proc/tty/driver/serial (if you have it). You should also be able to find it with the "setserial" command for `ttyS2`. But due to a bug (reported by me in Nov. 2000) you get the same "... busy" error message when you try this with "setserial".

To try to resolve this problem reboot or: exit or gracefully kill all likely conflicting processes. If you reboot: 1. Watch the boot−time messages for the serial ports. 2. Hope that the file that runs "setserial" at boot−time doesn't (by itself) create the same conflict again.

If you think you know what IRQ say `ttyS2` is using then you may look at /proc/interrupts to find what else (besides another serial port) is currently using this IRQ. You might also want to double check that any suspicious IRQs shown here (and by "setserial") are correct (the same as set in the hardware). A way to test whether or not it's a potential interrupt conflict is to set the IRQ to 0 (polling) using "setserial". Then if the busy message goes away, it was likely a potential interrupt conflcit. It's not a good idea to leave it permanently set at 0 since it will make the CPU work too hard.

15.19 "Input/output error" from setserial or stty

You may have typed "ttys" instead of "ttyS". You will see this error message if you try to use the setserial command for any device that is not a serial port. It also may mean that the serial port is in use (busy or opened) and thus the attempt to get/set parameters by setserial or stty failed. It could also mean that there isn't any serial port at the IO address that setserial thinks your port is at.

15.20 Overrun errors on serial port

This is an overrun of the hardware FIFO buffer and you can't increase its size. See "Higher Serial Thruput" in the Serial−HOWTO.

15.21 Modem doesn't pick up incoming calls

This paragraph is for the case where a modem is used for both dial−in and dial−out. If the modem generates a DCD (=CD) signal, some programs (but not mgetty) will think that the modem is busy. This will cause a problem when you are trying to dial out with a modem and the modem's DCD or DTR are not implemented correctly. The modem should assert DCD only when there is an actual connection (ie someone has dialed in), not when `getty` is watching the port. Check to make sure that your modem is configured to only assert DCD when there is a connection (&C1). DTR should be on (asserted) by the communications program whenever something is using, or watching the line, like `getty`, `kermit`, or some other comm program.

15.22 Troubleshooting Tools

These are some of the programs you might want to use in troubleshooting:

- "lsof /dev/ttyS*" will list serial ports which are open.
- "setserial" shows and sets the low−level hardware configuration of a port (what the driver thinks it is). See What is Setserial
- "stty" shows and sets the configuration of a port (except for that handled by "setserial"). See the Serial−HOWTO section: "Stty".
- "modemstat" or "statserial" will show the current state of various modem signal lines (such as DTR, CTS, etc.)
- "irqtune" will give serial port interrupts higher priority to improve performance.
- "hdparm" for hard−disk tuning may help some more.
- "lspci" shows the actual IRQs, etc. of hardware on the PCI bus.
- "pnpdump −−dumpregs" shows the actual IRQs, etc. of hardware for PnP devices on the ISA bus.
- Some "files" in the /proc tree (such as ioports, interrupts, and tty/driver/serial).

---

16. Flash Upgrades

Many modems can be upgraded by reprogramming their flash memories with an upgrade program which you get from the Internet. By sending this "program" from the PC via the serial port to the modem, the modem will store this program in its non−volatile memory (it's still there when the power is turned off). The instructions on installing it are usually on how to do in under Windows so you'll need to figure out how to do the equivalent under Linux (unless you want to install the upgrade under Windows). Sending the program to the modem is often called a download.

If the latest version of this HOWTO still contains this request (see New Versions of this HOWTO) please send me your experiences with installing such upgrades that will be helpful to others.

Here's the general idea of doing an upgrade. First, there may be a command that you need to send your modem to tell it that what follows is a flash ROM upgrade. In one case this was AT** You can do this by starting a communications program (such as minicom) and type. First type AT <enter> to see if your modem is there and answers "OK".

Next, you need to send an file (sometimes two files) directly to the modem. Communication programs (such as minicom) often use zmodem or kermit to send files to the modem (and beyond) but these put the file into packets which append headers and you want the exact file sent to the modem, not a modified one. But the kermit communications program has a "transmit" command that will send the file directly (without using the kermit packets) so this is one way to send a file directly. Minicom didn't have this feature in 1998.

Another way to send the file(s) would be to escape from the communications program to the shell (in minicom this is ^AJ) and then: `cat upgrade_file_name > /dev/ttyS2` (if your serial port is ttyS2). Then go back to the communication program (type fg at the command line prompt in minicom) to see what happened.

Here's an example session for a certain Rockwell modem (C−a is ^A):

```
- Run minicom
- Type AT** : see "Download initiated ..."
```

```
- C-a J
- cat FLASH.S37 > /dev/modem
- fg : see "Download flash code ..."
- C-a J
- cat 283P1722.S37 > /dev/modem
- fg : see "Device successfully programmed"
```

17. Other Sources of Information

17.1 Misc

- man pages for: `agetty(8)`, `getty(1m)`, `gettydefs(5)`, `init(1)`, `isapnp(8)`, `login(1)`, `mgetty(8)`, `setserial(8)`
- Your modem manual (if it exists). Some modems come without manuals.
- Serial Suite by Vern Hoxie is a collection of blurbs about the care and feeding of the Linux serial port plus some simple programs.
- The Linux serial mailing list. To subscribe, send email to `majordomo@vger.rutgers.edu`, with ``subscribe linux-serial'' in the message body. If you send ``help'' in the message body, you get a help message. The server also serves many other Linux lists. Send the ``lists'' command for a list of mailing lists.

17.2 Books

I've been unable to find a good up−to−date book on modems.

- The Complete Modem Reference by Gilbert Held, 1997. Contains too much info about obsolete topics. More up−to−date info may be found on the Internet.
- Modems For Dummies by Tina Rathbone, 1996. (Have never seen it.)
- The Modem Technical Guide by Douglas Anderson, 1996.
- Ultimate Modem Handbook by Cass R. Lewart, 1998.

17.3 HOWTOs

- Cable−Modem mini−howto
- ISDN Howto (not a LDP Howto) http://sdb.suse.de/sdb/en/html/isdn.html: drivers for ISDN "Modems". Much related info on this is in German.
- Linux−Modem−Sharing mini−howto. Computers on a network share a single modem for dial−out (like a shared printer).
- Modems−HOWTO: In French (Not used in creating this Modem−HOWTO)
- NET−3−4−HOWTO: all about networking, including SLIP, CSLIP, and PPP
- PPP−HOWTO: help with PPP including modem set−up
- Serial−HOWTO has info on Multiport Serial Cards used for both terminals and banks of modems. Covers the serial port in more detail than in the HOWTO.
- Serial−Programming−HOWTO: for some aspects of serial−port programming
- Text−Terminal−HOWTO: (including connecting up with modems)
- UUCP−HOWTO: for information on setting up UUCP

17.4 Usenet newsgroups

* comp.os.linux.answers FAQs, How−To's, READMEs, etc. about Linux.
* comp.os.linux.hardware Hardware compatibility with the Linux operating system.
* comp.os.linux.setup Linux installation and system administration.
* comp.dcom.modems Modems for all OS's

17.5 Web Sites

* Modem List of modems which work/don't_work under Linux
  http://www.idir.net/~gromitkc/winmodem.html
* Linux Serial Driver home page Includes info about support for PCI modems.
* Hayes AT modem commands Technical Reference for Hayes (tm) Modem Users
* AT Command Set and Register Summary for Analog Modem Modules (Cisco)
* Controlling your Modem with AT Commands
* Modem FAQs:
  Navas 28800−56K Modem FAQ
* Curt's High Speed Modem Page
* Much info on 56k modems 56k Modem = v.Unreliable
* Links to modem manufacturers
* More Links to modem manufacturers
* Identifying modems by FCC ID

---

18. Appendix A: How Analog Modems Work (technical) (unfinished)

18.1 Modulation Details

Intro to Modulation

This part describes the modulation methods used for conventional modems. It doesn't cover the high speed methods (modulus conversion) sometimes used by 56k Modems (v.90). But 56k modems also use the modulation methods described here.

Modulation is the conversion of a digital signal represented by binary binary (0 or 1) into an analog signal something like a sine wave. The modulated signal consists pure sine wave "carrier" signal which is modified to convey information. A pure carrier sine wave, unchanging in frequency and voltage, provides no flow of information at all (except that a carrier is present). To make it convey information we modify (or modulate) this carrier. There are 3 basic types of modulation: frequency, amplitude, and phase. They will be explained next.

Frequency Modulation

The simplest modulation method is frequency modulation. Frequency is measured in cycles per second (of a sine wave). It's the count of the number of times the sine wave shape repeats itself in a second. This is the same as the number of times it reaches it peak value during a second. The word "Hertz" (abbreviated Hz) is used to mean "cycles per second".

A simple example of frequency modulation is where one frequency means a binary 0 and another means a 1. For example, for some obsolete 300 baud modems 1070 Hz meant a binary 0 while 1270 Hz meant a binary 1. This was called "frequency shift keying". Instead of just two possible frequencies, more could be used to allow more information to be transmitted. If we had 4 different frequencies (call them A, B, C, and D) then each frequency could stand for a pair of bits. For example, to send 00 one would use frequency A. To send 01, use frequency B; for 10 use C; for 11 use D. In like manner, by using 8 different frequencies we could send 3 bits with each shift in frequency. Each time we double the number of possible frequencies we increase the number of bits it can represent by 1.

Amplitude Modulation

Once one understands frequency modulation example above including the possibilities of representing a few bits by a single shift in frequency, it's easier to understand both amplitude modulation and phase modulation. For amplitude modulation, one just changes the height (voltage) of the sine wave analogous to changing the frequency of the sine wave. For a simple case there could only be 2 allowed amplitude levels, one representing a 0−bit and another representing a 1−bit. As explained for the case of frequency modulation, having more possible amplitudes will result in more information being transmitted per change in amplitude.

Phase Modulation

To change the phase of a sine wave at a certain instant of time, we stop sending this old sine wave and immediately begin sending a new sine wave of the same frequency and amplitude. If we started sending the new sine wave at the same voltage level (and slope) as existed when we stopped sending the old sine wave, there would be no change in phase (and no detectable change at all). But suppose that we started up the new sine wave at a different point on the sine wave curve. Then there would likely be a sudden voltage jump at the point in time where the old sine wave stopped and the new sine wave began. This is a phase shift and it's measured in degrees (deg.) A 0 deg. (or a 360 deg.) phase shift means no change at all while a 180 deg. phase shift just reverses the voltage (and slope) of the sine wave. Put another way, a 180 deg. phase shift just skips over a half−period (180 deg.) at the point of transition. Of course we could just skip over say 90 deg. or 135 deg. etc. As in the example for frequency modulation, the more possible phase shifts, the more bits a single shift in phase can represent.

Combination Modulation

Instead of just selecting either frequency, amplitude, or phase modulation, we may chose to combine modulation methods. Suppose that we have 256 possible frequencies and thus can send a byte (8 bits) for each shift in frequency (since 2 to the 8 power is 256). Suppose also that we have another 256 different amplitudes so that each shift in amplitude represents a byte. Also suppose there are 256 possible phase shifts. Then a certain points in time we may make a shift in all 3 things: frequency, amplitude and phase. This would send out 3 bytes for each such transition.

No modulation method in use today actually does this. It's not practical due to the relatively long time it would take to detect all 3 types of changes. The main problem is that frequent shifts in phase can make it appear that a shift in frequency has happened when it actually didn't.

To avoid this difficulty one may simultaneous change only the phase and amplitude (with no change in frequency). This is called phase−amplitude modulation. It is also called quadrature amplitude modulation (= QAM) since there were only 4 possible phases (quadrature) in early versions of it. This method is used today for the common modem speeds of 14.4k, 28.8k, and 33.6k. The only significant case where this modulation method is not used today is for 56k modems. But even 56k modems exclusively use QAM (phase−amplitude modulation) in the direction from your PC out the telephone line. Sometimes even the other direction will also fall back to QAM when line conditions are not good enough. Thus QAM (phase−amplitude modulation) still remains the most widely used method on ordinary telephone lines.

18.2 56k Modems (v.90)

The "modulation" method used above 33.6k is entirely different than the common phase−amplitude modulation. Since ordinary telephone calls are converted to digital signals at the local offices of the telephone company, the fastest speed that you can send digital data by an ordinary telephone call is the same speed that the telephone company uses over its digital portion of the phone call transmission. What is this speed? Well, it's close to 64kbps. It would be 64k but sometimes bits are "stolen" for signalling purposes. But if the phone Co. knows that the link is not for voice, bits may not get stolen. The case of 64k will be presented and then it will be explained why the actual speed is lower (56k or less −−usually significantly less).

Thus 64k is the absolute top speed possible for an ordinary telephone call using the digital portion of the circuit that was designed to send digital encodings of the human voice. In order to use 64k, the modem must know exactly how the telephone company is doing its digital encoding of the analog signals. This task is far too complicated if both sides of a telephone call have only an analog interface to the telephone company. But if one side has a digital interface, then it's possible (at least in one direction). Thus if your ISP has a digital interface to the phone company, the ISP may send out a certain digital signal over the phone lines toward your PC. The digital signal from the ISP gets converted to analog at the local telephone office near your PC's location (perhaps near your home). Then it's your modem's task to try to figure out exactly what that digital signal was. If it could do this then transmission at 64k (the speed of the telephone company's digital signal) is possible in this direction.

What method does the telephone company use to digitally encode analog signals? It uses a method of sampling the amplitude of the analog signal at a rate of 8000 samples per second. Each sample amplitude is encoded as a 8−bit (ASCII−like) byte. (Note: 8 x 8000 = 64k) This is called "Pulse Code Modulation" = PCM. These bytes are then sent digitally on the telephone company's digital circuits where many calls share a single circuit using a time−sharing scheme known as "time division multiplexing". Then finally at a local telephone office near your home, the digital signal is de−multiplexed resulting in the same digital signal as was originally created by PCM. Then this signal is converted back to analog and sent to your home. Each 8−bit byte creates a certain amplitude of the analog signal. Your modem's task is to determine just what that PCM 8−bit byte was based on the analog amplitude it detects.

This is (sort of) "amplitude demodulation" but not really. It's not amplitude demodulation because there is no carrier. Actually, it's called "modulus conversion" which is the inverse of PCM. In order to determine the digital codes the telephone Co. used to create the analog signal, the modem must sample this analog signal amplitude at exactly the same points in time the phone Co. used when it created the analog signal. To do this a timing signal is generated from a residual 4kHz signal on the analog phone line. The creation of amplitudes to go out to your home/office at 8k amplitudes/sec sort of creates a 4kHz signal. Suppose every other amplitude was of opposite polarity. Then there would be a 4kHz sine−like wave created. Each amplitude is in a sense a 8−bit symbol and when to sample amplitudes is known as "symbol timing".

Now the encoding of amplitudes in PCM is not linear. At low amplitudes an increment of 1 in the PCM byte represents a much smaller increment (delta) in analog signal amplitude than would be the case if the amplitude being sampled were much higher. Thus for low amplitudes it's difficult to distinguish between adjacent byte values. To make it easier to do this (for 56k modems) certain PCM codes representing very low amplitudes are not used. This gives a larger delta between possible amplitudes and makes correct detection of them by your modem easier. Thus half the amplitude levels are not used by v.90. This is tantamount to each symbol (allowed amplitude level) representing 7 bits instead of 8. This is where 56k comes from: 7 bits/symbol x 8k symbols/sec = 56k bps. Of course each symbol is actually generated by 8−bits but only 128 bytes of the possible 256 bytes are actually used. There is a code table mapping these 128 8−bit bytes to 128 7−bit bytes.

But it's a little more complicated that this. If the line conditions are not nearly perfect, then even fewer possible levels (symbols) are used resulting in speeds under 56k. Also due to US government rules prohibiting high power levels on phone lines, certain high amplitudes levels can't be used resulting in only about 53.3k at best for "56k" modems.

Note that the digital part of the telephone network is bi−directional. Two such circuits are used for a phone call, one in each direction. The 56k signal is only used in one of these directions: from your ISP to your PC. The other direction, from your home/office to the ISP, uses the conventional phase−amplitude modulation scheme with a maximum of 36.6kbps (and not 53.3kbps). The analog portion of the circuit from your home/office to the nearest telephone Co. office was never intended to be bi−directional since it's only a single twisted pair. But due to sophisticated cancellation methods it's able to convey data simultaneously in both directions as explained in the next subsection.

18.3 Full Duplex on One Circuit

Modern modems are able to both send and receive signals simultaneously. One could call this "bidirectional" or "full duplex". This was once done by using one frequency for sending and another for receiving. Today, the same frequency is used for both sending and receiving. How this works is not easy to comprehend.

Most of the telephone system "main lines" are digital with two channels in use when you make a telephone call. What you say goes over one digital channel and what the other person says goes over the other (reverse) digital channel. Unfortunately, the part of the telephone system which goes to homes (and many offices) is not digital but only a single analog channel. If both modems were directly connected to the digital part of the phone system then bidirectional communication (sending and receiving at the same time) would be no problem because two channels would be available.

But the end portion of the signal path goes over just one circuit. How can there be two−way communication on it simultaneously? It works something like this. Suppose your modem is receiving a signal from the other modem and is not transmitting. Then there's no problem. But if your modem were to start transmitting (with the other received signal still flowing into your modem) it would drown out the received signal. If the transmitted signal was a "solid" voltage wave applied to the end of the line then there is no way any received signal could be present at that point.

But the transmitter has "internal impedance" and the transmitted signal applied to the end of the line is not solid (or strong enough) to completely eliminate the received signal coming from the other end. Thus while the voltage at the end of the line is mostly the stronger transmitted signal a small part of it is the desired received signal. All that is needed is to filter out this stronger transmitted signal and then what remains will be the signal from the other end which we want. To do this, one only needs to get the pure transmitted signal directly from the transmitter (before it's applied to the line) amplify it a determined amount, and then subtract it from the total signal present at the end of the line. Doing this in the receiver circuits leaves a signal which mostly came from the other end of the line.

18.4 Echo Cancellation

A signal traveling down a line in one direction may encounter changes in the line that will cause part of the signal to echo back in the opposite direction. Since the same circuit is used for bi−directional flow of data such echos will result in garbled reception. One way to ameliorate this problem is to send training signals once in a while to determine the echo characteristic of the line. This will enable one to predict the echos that will be generated by any given signal. Then this prediction method is used to predict what echos the transmitted signal will cause. Then this predicted echo signal is subtracted from the received signal. This cancels out the echoes.

19. Appendix B: Digital Modem Signal Processing (not done)

20. Appendix C: "baud" vs. "bps"

20.1 A simple example

``baud'' and ``bps'' are perhaps one of the most misused terms in the computing and telecommunications field. Many people use these terms interchangeably, when in fact they are not! bps is simply the number of bits transmitted per second. The baud rate is a measure of how many times per second a signal changes (or could change). For a typical serial port a 1−bit is −12 volts and a 0−bit is +12 v (volts). If the bps is 38,400 a sequence of 010101... would also be 38,400 baud since the voltage shifts back and forth from positive to negative to positive, etc. and there are 38,400 shifts per second. For another sequence say 111000111... there will be fewer shifts of voltage since for three 1's in sequence the voltage just stays at −12 volts yet we say that its still 38,400 baud since there is a possibility that the number of changes per second will be that high.

Looked at another way, put an imaginary tic mark separating each bit (even though the voltage may not change). 38,400 baud then means 38,400 tic marks per second. The tic marks at at the instants of permitted change and are actually marked by a synchronized clock signal generated in the hardware but not sent over the external cable.

Suppose that a "change" may have more than the two possible outcomes of the previous example (of +− 12 v). Suppose it has 4 possible outcomes, each represented by a unique voltage level. Each level may represent a pair of bits (such as 01). For example, −12v could be 00, −6v 01, +6v 10 and +12v 11. Here the bit rate is double the baud rate. For example, 3000 changes per second will generate 2 bits for each change resulting in 6000 bits per second (bps). In other words 3000 baud results in 6000 bps.

20.2 Real examples

The above example is overly simple. Real examples are more complicated but based on the same idea. This explains how a modem running at 2400 baud, can send 14400 bps (or higher). The modem achieves a bps rate greater than baud rate by encoding many bits in each signal change (or transition). Thus, when 2 or more bits are encoded per baud, the bps rate exceeds the baud rate. If your modem−to−modem connection is at 14400 bps, it's going to be sending 6 bits per signal transition (or symbol) at 2400 baud. A speed of 28800 bps is obtained by 3200 baud at 9 bits/baud. When people misuse the word baud, they may mean the modem speed (such as 33.6k).

Common modem bps rates were formerly 50, 75, 110, 300, 1200, 2400, 9600. These were also the bps rates over the serial_port−to−modem cables. Today the bps modem−to−modem (maximum) rates are 14.4k, 28.8k, 33.6k, and 56k, but the common rates over the serialPort−to−modem cables are not the same but are: 19.2k, 38.4k, 57.6k, 115.2k, 230.4k. The high speed of 230.4k is (as of late 2000) unfortunately not provided by most new (and old) hardware. Using modems with V.42bis compression (max 4:1 compression), rates up to 115.2k bps are possible for 33.6k modems. 203.2k (4 x 53.3k) is possible for 56k modems.

Except for 56k modems, most modems run at 2400, 3000, or 3200 baud. Even the 56k modems use these bauds for transmission and sometimes fall back to them for reception. Because of the bandwidth limitations on voice−grade phone lines, baud rates greater than 2400 are harder to achieve, and only work under conditions of good phone line quality.

How did this confusion between bps and baud start? Well, back when antique low speed modems were high speed modems, the bps rate actually did equal the baud rate. One bit would be encoded per phase change. People would use bps and baud interchangeably, because they were the same number. For example, a 300 bps modem also had a baud rate of 300. This all changed when faster modems came around, and the bit rate exceeded the baud rate. ``baud" is named after Emile Baudot, the inventor of the asynchronous telegraph printer. One way this problem gets resolved is to use the term "symbol rate" instead of "baud" and thus avoid using the term "baud". However when talking about the "speeds" between the modem and the serial port (DTE speed) baud and the symbol rate are the same. And even "speed" is a misnomer since we really mean flow rate.

21. Appendix D: Terminal Server Connection

This section was adapted from Text−Terminal−HOWTO.

A terminal server is something like an intelligent switch that can connect many modems (or terminals) to one or more computers. It's not a mechanical switch so it may change the speeds and protocols of the streams of data that go thru it. A number of companies make terminal servers: Xyplex, Cisco, 3Com, Computone, Livingston, etc. There are many different types and capabilities. Another HOWTO is needed to compare and describe them (including the possibility of creating your own terminal server with a Linux PC). Most are used for modem connections rather than directly connected terminals.

One use for them is to connect many modems (or terminals) to a high speed network which connects to host computers. Of course the terminal server must have the computing power and software to run network protocols so it is in some ways like a computer. The terminal server may interact with the user and ask what computer to connect to, etc. or it may connect without asking. One may sometimes send jobs to a printer thru a terminal server.

A PC today has enough computing power to act like a terminal server except that each serial port should have its own hardware interrupt. PC's only have a few spare interrupts for this purpose and since they are hard−wired you can't create more by software. A solution is to use an advanced multiport serial card which has its own system of interrupts (or on lower cost models, shares one of the PC's interrupts between a number of ports). See Serial−HOWTO for more info. If such a PC runs Linux with getty running on many serial ports it might be thought of as a terminal server. It is in effect a terminal server if it's linked to other PC's over a network and if its job is mainly to pass thru data and handle the serial port interrupts every 14 (or so) bytes. Software called "radius" is sometimes used.

Today real terminal servers serve more than just terminals. They also serve PC's which emulate terminals, and are sometimes connected to a bank of modems connected to phone lines. Some even include built−in modems. If a terminal (or PC emulating one) is connected directly to a modem, the modem at the other end of the line could be connected to a terminal server. In some cases the terminal server by default expects the callers to use PPP packets, something that real text terminals don't generate.

22. Appendix E: Other Types of Modems

This HOWTO currently only deals with the common type of modem used to connect PC's to ordinary analog telephone lines. There are various other types of modems, including devices called modems that are not really modems.

22.1 Digital−to−Digital "Modems"

The standard definition of a modem is sometimes broadened to include "digital" modems. Today direct digital service is now being provided to many homes and offices so a computer there sends out digital signals directly (well almost) into the telephone lines. But a device is still needed to convert the computer digital signal into the type allowed on telephone circuits and this device is sometimes called a modem. This HOWTO doesn't cover such modems but some links to documents that do may be found at the start of this HOWTO. The next 3 sections: ISDN, DSL and 56k, concern digital−to−digital "modems".

22.2 ISDN "Modems"

The "modem" is really a Terminal Adapter (TA). A Debian package "isdnutils" is available. There is a ISDN Howto in German with an English translation: http://sdb.suse.de/sdb/en/html/isdn.html. It's put out by the SuSE distribution of Linux and likely is about drivers available in that distribution. There is an isdn4linux package and a newsgroup: de.alt.comm.isdn4linux. Many of the postings are in German. You might try using a search engine (such as DejaNews) to find "isdn4linux".

22.3 Digital Subscriber Line (DSL)

DSL uses the existing twisted pair line from your home (etc.) to the local telephone office. This can be used if your telephone line can accept higher speeds than an ordinary modem (say 56k) sends over it. It replaces the analog−to−digital converter at the local telephone office with a converter which can accept a much faster flow of data (in a different format of course). The device which converts the digital signals from your computer to the signal used to represent digital data on the local telephone line is also called a modem.

22.4 56k Digital−Modems

For any 56k modem to work as a 56k modem in your home or office the other end must be connected directly to the digital system of the telephone company. Thus ISPs at the other end of the line must obtain special digital modems to provide customers with 56k service. There's more to it than this since banks of many modems are multiplexed onto a high capacity telephone cable that transports a large number of phone calls simultaneously (such as a T1, E1, ISDN PRI, or better line). This requires a concentrator or "remote access server". This has usually been done by stand−alone units (like PC's but they cost much more and have proprietary OSs). Now there are some cards one may insert into a PC's PCI bus to do this.

22.5 Leased Line Modems

These are analog and not digital modems. These special modems are used on lines leased from the telephone company or sometimes on just a long direct wire hookup. Ordinary modems for a telephone line will not normally work on such a line. An ordinary telephone line has about 40−50 volts (know as the "battery) on it when not in use and the conventional modem uses this voltage for transmission. Furthermore, the telephone company has special signals indicating a ring, line busy, etc. Conventional modems expect and respond to these signals. Connecting two such modems by a long cable will not provide the telephone signals on the cable and thus the modems will not work.

A common type of leased line used two pairs of wires (one for each direction) using V.29 modulation at 9600 baud. Some brands of leased line modems are incompatible with other brands.

---

23. Appendix F: Fax pixels (dots)

Here's some info on the bloated bandwidth required for standard fax including the dot density. You can of course send a fax via your modem if you dial the real telephone number of the recipient.

```
A4 paper:    216mm (horizontal) * 297mm (vertical)
normal mode       8dots/mm      * 3.85dots/mm
fine   mode                     * 7.7dots/mm
extra fine mode                 *15.4dots/mm
```

Each dot is either white or black and thus 1 bit. One sheet of A4 paper using fine mode is (216*8) * (297*7.7) = about 4 million dots. With a compression ratio of 8:1 it takes about 50 seconds at 9600bps for transmission.

---

24. Appendix G: Antique Modems

This section is about modems with modem−to−modem speeds of 9600 bps or less. Around 1980 many modems only had a speed of 300 bps (which was also 300 baud). This is only 0.3kbps. Modern modems are over 100 times faster. Many old−slow modems are still in use so they are not really "antique" quite yet. This section compares the antique modems with the modern ones. You should read it if you are interested in modem history or are intending to actually use an antique modem.

24.1 Autobauding

Various meanings

This term has a few different meanings. In general it means the automatic adjustment of modem−to−modem speed or modem−to−serial_port speed.

Modem−to−modem speed

Modern modems negotiate the modem−to−modem speed and protocol when they first connect to each other. If one side can't negotiate, the other side should accept whatever speed and protocol that the antique modem has available. Sometimes this is called "autobauding". When both modems automatically lower their speed due to a noisy line it's called "fallback". Thus users (or computer programs in your PC) don't need to deal with this (unless the S37 register has been set so as to disable autobauding).

But many old modems didn't have such autobauding (although many had fallback). If you have such a modem, it will likely work OK if the other modem you connect to is a modern one that can adjust it's speed and protocol to yours. But a problem arises if both modems which want to communicate with each other are both antique and don't support autobauding. How was this done?

In olden days, a computer dial−in site might have a number of phone lines, each of which would have a specific speed modem on it. For example, if you had a 1200 bps modem then you simply only dialed in to certain telephone numbers that supported that speed. Once a site obtained modems that could support various speeds on the same modem and automatically detect the callers speed (do autobauding) then people could call in using modems that didn't do this autobauding (providing that their speed and protocol was supported).

Modem−to−serial_port speed

When a modem modem is sent an init string (or a dial command), the modem detects the speed of the serial port and sets it's modem−to−serial_port speed to this value. It does this by sensing the speed of the "AT" at the beginning of the string.

Old modems couldn't do this and one would need to set the computer's serial port speed (with stty or the like) to the same exact value as the modem−to−modem speed (such as 1200 bps). If the modem had a choice of speeds one could use the AT register S37 to select one. But for dial−in when there was a choice of speeds (via autobauding), if a connection was made at say 2400 bps, then the modem−to−serial_port speed would change to 2400 bps. Then one would need to start getty at 2400 bps. Thus getty needed to adjust to whatever speed was coming from the modem. Today it's the other way around: the modem adjusts its modem−to−serial_port speed to whatever the serial port is using.

In Linux, there's a problem if the speed is set to a speed not supported by Linux's serial port (for example 7200 bps). You may dial out and connect at 7200 bps (both modem−to−modem and modem−to−serial_port speed) but you only see garbage since Linux doesn't support 7200 on the serial port. Once you connect there is no simple way to hang up because even the +++ escape sequence can't be sent to the modem over a 7200 baud interface.

To dial out by the antique method using a modern modem set &Q0 N0 and S17=5 (for 1200 bps). Some of the S17 settings vary with the make of modem except that S17=0 is the default that connects the modern way at the highest speed supported.

The original getty program configuration file /etc/gettydefs has an optional autobauding feature (see "next−label" in the manual) which will change the speed of the serial port. The agetty program has a similar baud rate detection feature. This was used to adjust the speed of the serial port to the modem−to−modem speed of an incoming dial−in call.

Modern modems can use almost any serial port speed. To do this they employ speed buffering and flow control. Speed buffering means that modems have buffers so that there can be a difference between the modem−to−modem speed and the modem−to−serial_port speed. If the flow entering the modem is faster than the flow exiting it, the excess flow is simply stored in a buffer in the modem. Then to prevent the buffer from overflowing, the modem sends a flow control signal to stop the input flow to the modem. This is true for either direction of flow. See Flow Control for more details.

24.2 Before AT Commands

Hayes introduced the AT command set and other modem manufacturers adopted it as a standard. Before the AT commands, many modems used dip switches to configure the modem. Another command set is the CCITT V.25bis command set. Some modems supported both CCITT and AT commands. The CCITT V.25bis also specifies how Synchronous modem−to−serial_port communication is to take place using either the ASCII or 8−bit EBCDIC character sets.

24.3 Data Compression and Error Correction

MNP 2, 3, or 4 were used for error correction. MNP 5 was compression. Modern modems generally use V42 (error correction) and V42bis (compression). Many modems support both MNP and V42.

24.4 Obsolete CCITT (ITU) and Bell Protocols

- Bell 103 300 bps; frequency shift keying = FSK
- v.21 300 bps; frequency shift keying (used a different frequency than Bell 103)
- v.23 1200/75 bps and 600/75 bps asymmetric; 75 bps is the reverse channel; frequency shift keying
- Bell 212A 1200 bps; quadrature differential phase shift keying = QDPSK = DPSK
- v.22 1200 bps; fallback to 600 bps ; QDPSK = DPSK
- v.22bis 2400 bps; quadrature (QDPSK) amplitude modulation = QAM
- v.32 9600 bps; QAM
- v.32bis 14400 bps; QAM

END OF Modem−HOWTO