

CHAPTER 3

Installing Asterisk

I long to accomplish great and noble tasks, but it is my chief duty to accomplish humble tasks as though they were great and noble. The world is moved along, not only by the mighty shoves of its heroes, but also by the aggregate of the tiny pushes of each honest worker.

—Helen Keller


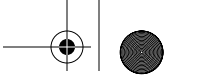
In the previous chapter, we discussed preparing a system to install Asterisk. Now it's time to obtain, extract, compile, and install the software.

Although a large number of Linux distributions* and PC architectures are excellent candidates for Asterisk, we have chosen to focus on a single distribution in order to maintain brevity and clarity throughout the book. The instructions that follow have been made as generic as possible, but you may notice a leaning toward Red Hat structures and utilities. We have chosen to focus on Red Hat because its command set, directory structure, and so forth are likely to be familiar to the majority of users (we have found that most Linux administrators are familiar with Red Hat, even if they don't prefer it). However, this doesn't mean that Red Hat is the only choice, or even the best one for you. A question that often appears on the mailing lists is: "Which distribution of Linux is the best to use with Asterisk?" The multitude of answers generally boils down to "the one you like the best."

What Packages Do I Need?

Asterisk uses three main packages: the main Asterisk program (*asterisk*), the Zapata telephony drivers (*zaptel*), and the PRI libraries (*libpri*). If you plan on a pure VoIP network, the only real requirement is the *asterisk* package. The *zaptel* drivers are



* And some non-Linux operating systems as well, such as Solaris, *BSD, and OS X. However, while people have managed to successfully run Asterisk on these alternative systems, Asterisk was, and continues to be, actively developed for Linux.



required if you are using analog or digital hardware, or if you're using the *ztdummy* driver (discussed later in this chapter) as a timing interface. The *libpri* library is technically optional unless you're using ISDN PRI interfaces, and you may save a small amount of RAM if you don't load it, but we recommend that it be installed in conjunction with the *zaptel* package for completeness.

One other package you may want to install is *asterisk-sounds*. While Asterisk comes with many sound prompts in the main source distribution, the *asterisk-sounds* package will give you even more. If you would like to expand the number of professionally recorded prompts for use with your Asterisk system, this package is essential. Some of our examples in the following chapters will make use of files included in this package, so we will assume that you have it installed.

Package Requirements



To compile Asterisk, you must install the *GCC compiler* (Version 3.x or later) and its dependencies. While Version 2.96 of GCC may work for the time being, future versions will not support it. Asterisk also requires *bison*, a parser generator program that replaces *yacc*, and *ncurses* for CLI functionality. The cryptographic library in Asterisk requires *OpenSSL* and its development packages. If you want to use *ztdummy* for timing, or any of the hardware drivers provided by Zaptel, you'll need to install the *zaptel* package as well. If you are installing *libpri*, be sure to install it before *asterisk* (see "Compiling libpri").

Zaptel requires *libnewt* and its development packages for the *ztool* program (see "Using ztcfg and ztool," below) and the *usb-uhci* module for *ztdummy*. If you're using PRI interfaces, Zaptel also requires the *libpri* package (again, even if you aren't using PRI circuits, we recommend that you install *libpri* along with *zaptel*).

The following sections discuss how to obtain, extract, compile, and install the *asterisk*, *zaptel*, *libpri*, and *asterisk-sounds* packages.

Obtaining the Source Code

The Asterisk source code can be obtained either through FTP or CVS. We will show you how to acquire the source with both methods, although you only need to use one of them to retrieve the packages (FTP is the preferred method).

Obtaining Asterisk Source Code from FTP

The Asterisk source code can be obtained from the Digium FTP server, located at *ftp://ftp.digium.com*. The easiest way to obtain the stable release is through the use of the program *wget*.

Stable and Head

Asterisk comes in two different flavors, generally referred to as *stable* and *head*. Stable, as the name implies, is the established branch of Asterisk for use in production systems. The head branch is what the developers use to test new features and bug fixes.

Bug fixes (not features) are merged over to the stable branch after a reasonable period of testing. It is entirely possible that the development branch may be broken at certain points during testing; thus, the stable branch is what you will want to run your production system on, and it is what we will be using throughout this book.

You can obtain stable releases via FTP. Both the stable and head branches of Asterisk can also be obtained from CVS, as explained later in this chapter. However, it is important to note the difference between *releases* and CVS. Releases are snapshots from the stable CVS tree, tagged with a version number and released via the FTP server when a new stable release is deemed ready. Note that the stable CVS branch is *not* a release—it's a work in progress, and it may be buggy (i.e., not so stable after all). The FTP tarballs are the actual releases.

To summarize, use only stable releases obtained via the FTP server for production systems.

Note that we will be making use of the `/usr/src/` directory to extract and compile the Asterisk source. Also be aware that you will need *root* access to write files to the `/usr/src/` directory and to install Asterisk and its associated packages.

To obtain the latest stable source code via `wget`, enter the following commands on the command line:

```
# cd /usr/src/  
# wget --passive-ftp ftp.digium.com/pub/asterisk/asterisk-1.*.tar.gz  
# wget --passive-ftp ftp.digium.com/pub/asterisk/asterisk-sounds-*.tar.gz  
# wget --passive-ftp ftp.digium.com/pub/zaptel/zaptel-*.tar.gz  
# wget --passive-ftp ftp.digium.com/pub/libpri/libpri-*.tar.gz
```



As long as Digium doesn't change the way they put things on the FTP site, the `wget` command will automagically get the latest version. You may also replace the wildcard mask (*) with the currently available software version.

Now that you've retrieved the files for Asterisk and the Digium hardware, you are ready to extract the code.

Extracting the Source Code

If you use `wget` to obtain the source code from the FTP server, you will need to extract it before compiling. If you didn't download the packages to `/usr/src/`, either

move them there now, or specify the full path to their location. We will be using the GNU *tar* application to extract the source code from the compressed archive. This is a simple process that can be achieved through the use of the following commands:

```
# cd /usr/src/  
# tar zxvf zaptel-*.tar.gz  
# tar zxvf libpri-*.tar.gz  
# tar zxvf asterisk-*.tar.gz  
# tar zxvf asterisk-sounds*.tar.gz
```

These commands will extract the packages and source code to their respective directories.

Obtaining Asterisk Source Code from CVS

The Concurrent Versioning System (CVS) is a tool that provides a central repository that large (and diverse) development teams can use to manage the multitude of files associated with a development project. When a change is made, it is committed to the CVS server, where it is immediately available for download and compilation. Another added benefit of using CVS is that the version for any particular file can be rolled back to a certain instance, so that if something was working at one point but a change causes it to break, you can easily revert to the working version. This is true for the entire tree as well. If you find that installing the latest version of Asterisk causes any part of the system to break, you can “roll back” to an earlier point in time and investigate the cause of the problem.

If you are a developer looking to obtain the latest updates to the source code, you will need to get them from the CVS servers. You can also download the stable branch via CVS:

- Export the *CVSROOT* path:

```
# cd /usr/src/  
# export CVSROOT=:pserver:anoncvs:anoncvs@cvs.digium.com:/usr/cvsroot
```
- Download *HEAD* from CVS:

```
# cvs checkout zaptel libpri asterisk
```
- Download *STABLE 1.0* from CVS:

```
# cvs checkout -r v1-0 zaptel libpri asterisk
```
- Download *STABLE 1.2* from CVS:

```
# cvs checkout -r v1-2 zaptel libpri asterisk
```
- Download optional modules from CVS:

```
# cvs checkout asterisk-sounds asterisk-addons
```

Again, note that the stable branch available from CVS is not a release and should not be used for production systems.

Compiling Zaptel

Figure 3-1 shows the layers of interaction between Asterisk and the Linux kernel with respect to hardware control. On the Asterisk side is the Zapata channel module, *chan_zap*. Asterisk uses this interface to communicate with the Linux kernel, where the drivers for the hardware are loaded.

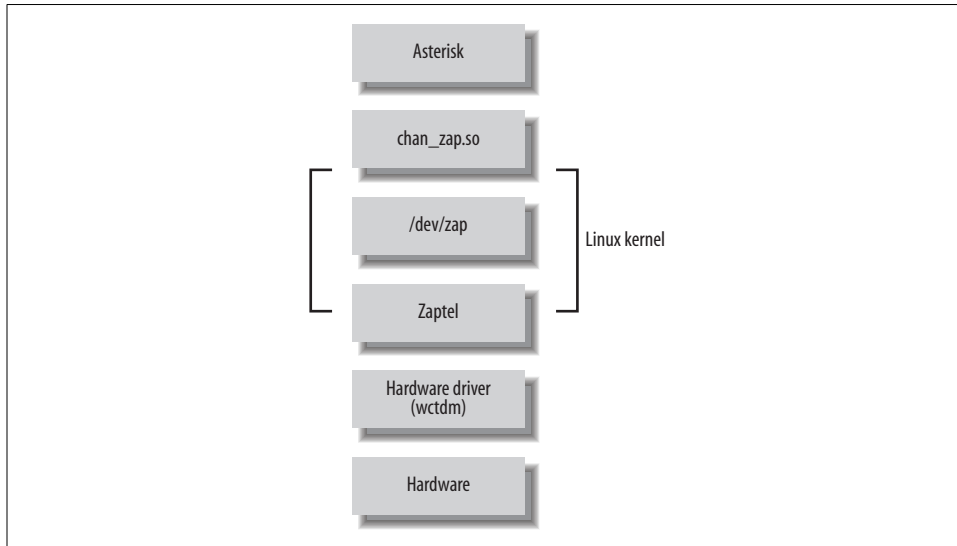


Figure 3-1. Layers of device interaction with Asterisk

The Zaptel interface is a kernel loadable module that presents an abstraction layer between the hardware drivers and the Zapata module in Asterisk. It is this concept that allows the device drivers to be modified without any changes being made to the Asterisk source itself. The device drivers are used to communicate with the hardware directly and to pass the information between Zaptel and the hardware.



While Asterisk itself compiles on a variety of platforms, the Zaptel drivers are Linux-specific—they are written to interface directly with the Linux kernel. There are no official Zaptel drivers for other operating systems, although work has been going on to write drivers for FreeBSD.

We will discuss the Zaptel compile-time options momentarily, in “The *zconfig.h* File.” First, let’s take a look at compiling and installing the drivers. (The configuration of Zaptel drivers will be discussed in the next chapter.)



Before compiling the Zaptel drivers on a system running a Linux 2.4 kernel, you should verify that `/usr/src/` contains a symbolic link named `linux-2.4` pointing to your kernel source. If the symbolic link doesn't exist, you can create it with the following command (assuming you've installed the source in `/usr/src/`):

```
# ln -s /usr/src/`uname -r` /usr/src/linux-2.4
```

Computers running Linux 2.6 kernel-based distributions do not usually require the use of the symbolic link, as these distributions will search for the kernel build directory automatically. However, if you've placed the build directory in a nonstandard place (i.e., somewhere other than `/lib/modules/<kernel version>/build/`), you will require the use of the symbolic link.

The ztdummy Driver

In Asterisk, certain applications and features require a timing device in order to operate (Asterisk won't even compile them if no timing device is found). All Digium PCI hardware provides a 1-kHz timing interface. If you lack the PCI hardware required to provide timing, the `ztdummy` driver can be used as a timing device. On Linux 2.4 kernel-based distributions, `ztdummy` must use the clocking provided by the UHCI USB controller. The driver looks to see that the `usb-uhci` module is loaded and that the kernel version is at least 2.4.5. Older kernel versions are incompatible with `ztdummy`.

On a 2.6 kernel-based distribution, `ztdummy` does not require the use of the USB controller. (As of v2.6.0, the kernel now provides 1-kHz timing with which the driver can interface; thus, the USB controller hardware requirement is no longer necessary.)

The default `Makefile` configuration does not create `ztdummy`. To compile `ztdummy`, you must remove a comment marker from the `Makefile`. Open it in your favorite text editor and look for the following line:

```
MODULES=zaptel tor2 torisa wcusbc wcfxo wctdm \  
ztdynamic ztd-eth wct1xxp wct4xxp wcte11xp # ztdummy
```

Remove the hash (#) symbol from in front of "ztdummy," save the file, and compile Zaptel as usual.

The Zapata Telephony Drivers

Compiling the Zapata telephony drivers for use with your Digium hardware is straightforward—simply run `make` for either the 2.4 or 2.6 Linux kernels (the `Make-`

* The # symbol is most widely known as "hash," so that is what we have chosen to call it. North Americans tend to call it a "pound sign," the ITU uses the term "square," and yet others call it a "crosshatch" or "number sign." Another term, made up by Don Macpherson to describe the # symbol during initial training on an early PBX system, is "octothorpe." This term eventually found its way into memos and letters at Bell Labs, then into other official documents, and from there leaked to the Internet.

file will determine the kernel version for you). Use these commands to compile Zap-*tel* (replace *version* with your version of *zaptel*):

```
# cd /usr/src/zaptel-version
# make clean
# make
# make install
```



While running `make clean` is not always necessary, it's a good idea to run it before recompiling any of the modules, as it will remove the compiled binary files from within the source code directory. You can also use it to clean up after installing, if you don't like to leave the compiled binaries floating around. Note that this removes the binaries only from the source directory, not from the system.

In addition to the executables, `make clean` also removes the intermediary files (i.e., the object files) after compilation. You don't need them occupying space on your hard drive.

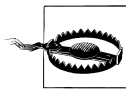
If you're using a system that makes use of the `/etc/rc.d/init.d/` or `/etc/init.d/` directories, you may wish to run the `make config` command as well. This will install the startup scripts and configure the system, using the `chkconfig` command to load the *zaptel* module automatically at startup.



The Debian equivalent of `chkconfig` is `update-rc.d`.

Using *ztcfg* and *zttool*

Two programs installed along with *Zaptel* are *ztcfg* and *zttool*. The *ztcfg* program is used to read the configuration in `/etc/zaptel.conf` to configure the hardware. The *zttool* program can be used to check the status of your installed hardware. For instance, if you are using a T1 card and there is no communication between the endpoints, you will see a red alarm. If everything is configured correctly and communication is possible, you should see an "OK." The *zttool* application is also useful for analog cards, because it tells you their current state (configured, off-hook, etc.). The use of these programs will be explored further in the next chapter.



The *libnewt* libraries and its development packages (*newt-devel* on Red Hat-based distributions) must be installed for *zttool* to be compiled.

The zconfig.h File

The *zconfig.h* file is where many of the Zaptel compile-time options lie. For the most part, you should not need to edit this file, but below are some of the options that may be of interest. To enable the options, remove the comment tags (*/* */*). If you decide to enable any of these options, be sure to do a `make clean` before recompiling and reinstalling Zaptel.

Boost ringer

By enabling the `BOOST_RINGER` option, you increase the amount of voltage supplied to a telephone during ringing from $\sim 70\text{V}$ to $\sim 89\text{V}$. Some devices may not detect ringing below certain voltages, so this setting may be necessary. Note that upping the voltage requires more power, and that it will probably only be necessary on a telephone connected to a long loop. Basically, you should leave this alone unless the far end isn't detecting ringing properly. To enable this option, uncomment the following line:

```
/* #define BOOST_RINGER */
```

The `BOOST_RINGER` option can also be declared when loading the driver via `modprobe`, so it does not need to be compiled into the driver (recommended).

Disable μ -law/A-law precomputation

Defining `CONFIG_CALC_XLAW` tells Zaptel to not precompute μ -law/A-law into tables and to recalculate it for each sample. We haven't timed it, but the original coder felt that if you have a small number of channels and/or a small level-2 cache, it may be quicker to execute the calculation code than to actually do a lookup on the table loaded into memory.

To enable this option, uncomment the following line within *zconfig.h*:

```
/* #define CONFIG_CALC_XLAW */
```

Enable MMX optimization

You can enable MMX optimization (if your processor supports it) by removing the comment tags around the following line:

```
/* #define CONFIG_ZAPTEL_MMX */
```

Be aware that `CONFIG_ZAPTEL_MMX` is considered to be incompatible with AMD processors and can cause system instability.

Choose echo cancellation method

All the echo cancellers in Asterisk use a Finite Impulse Response (FIR) algorithm. The differences between them—mostly in code implementation and slight algorithm

tweaks—are minimal. By default, the *MARK2* echo canceller is used, and it is generally considered the most robust. To change the default, add comment tags around the `#define ECHO_CAN_MARK2` line and uncomment another line:

```
/* #define ECHO_CAN_STEVE */
/* #define ECHO_CAN_STEVE2 */
/* #define ECHO_CAN_MARK */
#define ECHO_CAN_MARK2
/* #define ECHO_CAN_MARK3 */
```

Enable aggressive suppression

Aggressive residual echo suppression with the *MARK2* echo canceller can be enabled by removing the comment tags around the following line:

```
/* #define AGGRESSIVE_SUPPRESSOR */
```

The aggressive suppressor makes the nonlinear processor (NLP) stronger. What the NLP essentially does is say, “If the sample is that quiet anyway, make the volume level about 0.”

Disable echo cancellation

When echo cancellation is enabled in Asterisk, it is possible to disable it by sending a 2100-Hz tone at the beginning of a call. If you do not want Asterisk to disable echo cancellation even when it detects the echo cancel disable tone, uncomment the following line:

```
/* #define NO_ECHOCAN_DISABLE */
```

Fax machines and modems use the 2100-Hz tone during negotiation, and Asterisk monitors for this tone during call setup.

Enable HDLC

When using the Zaptel driver with T1 or E1 hardware, you can configure Zaptel to use TDM channels for data instead of voice. To enable HDLC functionality in the drivers, uncomment the following line:

```
/* #define CONFIG_ZAPATA_NET */
```

For this change to be meaningful, you must also use the *sethdlc* utility and perform some configuration in *zapata.conf*.

Enable ZapRAS

You can also make use of the *ZapRAS* program to turn Asterisk into a Remote Access Server (RAS) for use with your ISDN connections. To enable this functionality, you must uncomment the following line from within the *zconfig.h* file:

```
/* #define CONFIG_ZAPATA_PPP */
```

You must also patch Asterisk and configure a PPP daemon, so be aware that this task is nontrivial.

Enable Zaptel's watchdog

You can tell Zaptel to monitor the status of interfaces via its built-in “watchdog.” It will check if the interfaces stop taking interrupts or otherwise misbehave. If this happens, the hardware will automatically be restarted. To enable the watchdog, uncomment this line:

```
/* #define CONFIG_ZAPTEL_WATCHDOG */
```

Set default tone zone

The tone zone info option is used to select which set of tones (e.g., dial tone, busy indication, ring tone, stutter, etc.), as defined in the *zonedata.c* file, should be used as the default. The *zonedata.c* file contains the frequencies and patterns that Asterisk uses to communicate on the PSTN networks in various countries and to signal connected telephones. The default tone zone (0) is used to indicate North American signaling frequencies. Other tone zones include Australia (1), France (2), Japan (7), Taiwan (14), and many others. You can change the default on the following line:

```
#define DEFAULT_TONE_ZONE 0
```

Enable CAC ground start signaling

Some devices, such as the FXO ports on a Carrier Access Corporation (CAC) channel bank, have nonstandard FXS ground start signaling start states (A=low, B=low). You can configure the drivers to use this state by removing the comment tags around the following line:

```
/* #define CONFIG_CAC_GROUNDSTART */
```

TDM400P Revision H PCI ID workaround

If you happen to be using an older TDM400P Revision H card, you may find that it sometimes forgets its PCI ID. To make the *wctdm* driver essentially match all subvendor IDs, uncomment the following line:

```
/* #define TDM_REVH_MATCHALL */
```

This may be required when using older revisions of TDM400P cards with newer versions of Asterisk, due to a change in the subvendor ID code. This has been known to cause the following type of error when loading the *wctdm* module:

```
# ZT_CHANCONFIG failed on channel 12: No such device or address (6)
```

Uncommenting the `#define` line above should resolve this problem.

Passing Module Parameters to Configure Zaptel

Some of the Zaptel options can also be enabled when loading the module, by passing module parameters to the *wctdm* driver. You can list these parameters at load time (as opposed to statically changing them in the *zconfig.h* file) with the `modinfo` command:

```
# modinfo -p wctdm
debug int
loopcurrent int
robust int
_opermode int
opermode string
timingonly int
lowpower int
boostringer int
fxshonormode int
```

You then pass the module parameters to the `modprobe` command. For example, you can use the following command to activate the `boostringer` parameter when the module is loaded, instead of statically defining its use with `#define BOOST_RINGER` in the *zconfig.h* file:

```
# modprobe wctdm boostringer=1
```

Another common parameter to pass to a module is `opermode`. By passing `opermode` to the *wctdm* driver, you can configure the TDM400P to better deal with line impedances for your country. `opermode` accepts a two-letter country code as its argument.

Compiling libpri

Compiling and installing *libpri* follows the same pattern as described above for *zaptel*. *libpri* is used by various makers of Time Division Multiplexing (TDM) hardware, but even if you don't have the hardware installed it is safe to compile and install this library. You must compile and install *libpri* before Asterisk, as it will be detected and used when Asterisk is compiled. Here are the commands (replace *version* with your version of *libpri*):

```
# cd /usr/src/libpri-version
# make clean
# make
# make install
```

Compiling Asterisk

Once you've compiled and installed the *zaptel* and *libpri* packages (if you need them), you can move on to Asterisk. This section walks you through a standard installation and introduces some of the alternative `make` arguments that you may find useful. We'll also look at how you can edit the *Makefile* to optimize the compilation of Asterisk.

Standard Installation

Asterisk is compiled with *gcc* through the use of the GNU *make* program. Unlike many other programs, there is no need to run a configuration script for Asterisk. To get started compiling Asterisk, simply run the following commands (replace *version* with your version of Asterisk):

```
# cd /usr/src/asterisk-version
# make clean
# make
# make install
# make samples
```

Be aware that compile times will vary between systems. On a current-generation processor, you shouldn't need to wait more than five minutes. At Astricon, someone reported successfully compiling Asterisk on a 133-MHz Pentium, but it took approximately five hours. You do the math.

Run the `make samples` command to install the default configuration files. Installing these files (instead of configuring each file manually) will allow you to get your Asterisk system up and running much faster. Many of the default values are fine for Asterisk. Files that require editing will be explained in future chapters.



If you already have configuration files installed in `/etc/asterisk/` when you run the `make samples` command, `.old` will be appended to the end of each of your current configuration files—for example, `extensions.conf` will be renamed `extensions.conf.old`. Be careful, though, because if you run `make samples` more than once you will overwrite your original configuration files!

The sample configuration files can also be found in the `configs/` subdirectory within your Asterisk sources directory.

If you're using a system that makes use of the `/etc/rc.d/init.d/` or `/etc/init.d/` directories, you may wish to run the `make config` command as well. This will install the startup scripts and configure the system (through the use of the `chkconfig` command) to execute Asterisk automatically at startup.

Alternative make Arguments

There are several other `make` arguments that you can pass at compile time. While some of these will be discussed here, the remainder are used internally within the file and really have no bearing or use for the end user. (Of course, new functions may have been added, so be sure to check the *Makefile* for other options.)

Let's take a look at some useful `make` arguments.

make clean

The `make clean` command is used to remove the compiled binaries from within the source directory. This command should be run before you attempt to recompile or, if space is an issue, if you would like to clean up the files.

make update

This command is used to update the existing code from the Digium CVS server. If you downloaded the source code from the FTP server, you will receive an error stating so.



A common problem that you may find if you update with the `cvs update` command is that when you then do a `show version` at the Asterisk command-line interface (CLI), your version does not appear to have been updated. This problem can be resolved by removing the hidden `.version` file within the Asterisk source code directory before recompiling, or by using the `make update` command (which will remove the file for you).

make upgrade

If you run the `make install` command to install Asterisk after using the `make update` command to update from CVS, the `.version` file will not be updated. If you do not want to manually delete the `.version` file before running `make` and `make install`, you can use the `make upgrade` command instead.

make webvmail

The Asterisk Web Voicemail script is used to give a graphical interface to your voicemail account, allowing you to manage and interact with your voicemail remotely from a web browser.

When you run the `make webvmail` command, the Asterisk Web Voicemail script will be placed into the `cgi-bin/` directory of your HTTP daemon. If you have specific policies with respect to security, be aware that it uses a `setuid root` Perl script. This command will install only on a Red Hat or Fedora box, as other distributions may have different paths to their `cgi-bin/` directories. (This, of course, can be changed by editing the *Makefile*.)

make progdocs

This command will create documentation using the *doxygen* software from comments placed within the source code by the developers. You must have the appropriate *doxygen* software installed on your system in order for this to work. Note that *doxygen* assumes that the source code is well documented, which, sadly, is not always the case.

make mpg123

Asterisk uses the *mpg123* program to stream MP3s during the use of Music on Hold (MoH). Because Asterisk only works with *mpg123* v0.59r, this shortcut will determine if the correct version of *mpg123* is installed on your system and, if not, will attempt to download, extract, and compile it for you. Be aware that newer versions will not work, and some distributions even symbolically link *mpg321* and *mpg123*, which are entirely different programs. If you run the `make install` command after running this command, Asterisk will detect the directory and install it for you as well.

make config

The `make config` command will install Red Hat–style initialization scripts, if the */etc/rc.d/init.d* or */etc/init.d* directories are found to exist. If they do exist, the scripts are installed with file permissions equal to 755. If the script detects that */etc/rc.d/init.d* exists, the `chkconfig --add asterisk` command will also be run to cause Asterisk to be started automatically at boot time. This is not the case, however, with distributions that only use the */etc/init.d/* directory. Running `make config` will not do anything to an already running Asterisk process, or start one if it's not running.

This script currently is only really useful on a Red Hat–based system, although initialization scripts are available for other distributions (such as Gentoo, Mandrake, and Slackware) in the *./contrib/init.d/* directory of your Asterisk source directory.

Editing the Makefile

At the top of the *Makefile* contained within the Asterisk source directory are several options for optimizing the compilation of Asterisk. You can enable GSM codec optimizations (with the use of MMX instructions), disable configuration file overwrites, add extra debugging information, change Asterisk's installation and staging directories, and modify which type of processor you are compiling for. While you may never edit or require any of these options, they are mentioned here for completeness.

Enabling GSM optimizations

Uncomment the following line in your Asterisk *Makefile* to enable GSM codec optimizations on x86 CPU architectures that support MMX instructions:

```
#K6OPT = -DK6OPT
```

This includes newer Pentium processors, Pentium Pros, and the AMD K6 and K7 processors; however, you may not want to enable MMX support unless you have a true Intel processor, as problems have been reported with the MMX instructions on non-Intel processors.

Disabling configuration file overwrites

By default, Asterisk will overwrite your configuration files if you run `make samples` more than once. To change this behavior, change the `y` in the line below to `n`:

```
OVERWRITE=y
```

Enabling debug profiling information

Debug symbols allow you to do symbolic debugging. The profiling information (`-pg`) flag will produce a file when you run Asterisk that can be processed in order to obtain information about how long (relatively) Asterisk spends in each function. Use of the `-pg` flag is not recommended for a normal build, but it may be useful during development. To enable profiling information, replace the `-g` in the following line with `-pg`:

```
DEBUG=-g
```

Specifying where to install Asterisk after compiling

You can change the directory where Asterisk is installed by specifying a path on the following line:

```
INSTALL_PREFIX=
```

Changing the staging directory

The staging directory is where Asterisk temporarily copies its files during the install process. You may want the files to be copied to a directory such as `/tmp/asterisk/`. If no staging directory is specified (the default), Asterisk will use the source directory. To specify a staging directory, enter the desired directory on this line:

```
DESTDIR=
```

Compiling on VIA motherboards

On VIA-based motherboards, you need to set the processor to `i586`. If Asterisk detects the processor as `i686`, you may get random core dumps. To force Asterisk to compile using `i586`, remove the comment from the following `PROC` line in the *Makefile* (line 81, at the time of this writing):

```
# Pentium & VIA processors optimize  
# PROC=i586
```

Using Precompiled Binaries

While the documented process of installing Asterisk expects you to compile the source code yourself, there are Linux distributions (such as Debian) that include pre-compiled Asterisk binaries. Failing that, you may be able to install Asterisk with the package managers that those distributions of Linux provide (such as *apt-get* for

Debian and *portage* for Gentoo). However, you may also find that many of these pre-built binaries are quite out of date and do not follow the same furious development cycle as Asterisk.

Finally, there do exist basic, precompiled Asterisk binaries that can be downloaded and installed in whatever Linux distribution you have chosen. However, the use of precompiled binaries doesn't really save much time, and we have found that compiling Asterisk with each install is not a very cumbersome task. We believe that the best way to install Asterisk is to compile from the source code, so we won't discuss pre-built binaries very much in this book. In the next chapter, we'll look at how to initially configure Asterisk and several kinds of channels.

Installing Additional Prompts

The *asterisk-sounds* package contains many useful professionally recorded prompts. It is highly recommended that you install it now, as we will be using some of the prompts from this package in later chapters. To do so, run the following commands:

```
# cd /usr/src/asterisk-sounds
# make install
```

Other Useful Add-ons

The *asterisk-addons* package contains code to allow the storage of Call Detail Records (CDRs) to a MySQL database and to natively play MP3s, as well as an interpreter for loading Perl code into memory for the life of an Asterisk process. Programs are placed into *asterisk-addons* when there are licensing issues preventing them from being implemented directly into the Asterisk source code, or when they are not yet ready for primetime.

The *g729/* directory contains the code and registration program for the proprietary G.729 codec. Even if your end devices have the G.729 codec installed, in order to allow the phones to communicate with Asterisk using G.729 (e.g., in voicemail or to allow attended transfers), you must purchase a license. Licenses for the codec can be purchased online from Digium and activated with the registration program contained in the *g729/* directory.

Updating Your Source Code

Instead of deleting the sources and downloading the entire tree every time you want to update, you can update just the files that have changed since the last revision. To do this, change into the directory containing the files you want to update and run the `make update` command:


```
# cd /usr/src/asterisk/  
# make update  
# make clean  
# make upgrade
```

Note that this will work only with code obtained via the CVS method (see “make update,” earlier in this chapter). The `make upgrade` command is used only in the Asterisk source directory. In other directories, use `make install`.

Common Compiling Issues

There are many common compiling issues that users often run into. Here are some of the more common problems, and how to resolve them.

Asterisk

First, let’s take a look at some of the errors you may encounter when compiling Asterisk.

C compiler cannot create executables

If you receive the following error while attempting to compile Asterisk, you must install the `gcc` compiler and its dependencies:

```
checking whether the C compiler (gcc ) works... no  
configure: error: installation or configuration problem: C compiler cannot create  
executables.  
make: *** [editline/libedit.a] Error 1
```

The following packages are required for `gcc`:

- `gcc`
- `glibc-kernheaders`
- `cpp`
- `binutils`
- `glibc-headers`
- `glibc-devel`

These can be installed manually, by copying the files off of your distribution disks, or through the `yum` package manager, with the command `yum install gcc`.

bison: command not found

The following error may be encountered if the `bison` parser, which is required for parsing expressions in the `extensions.conf` file, is not found:

```
bison ast_expr.y -name-prefix=ast_yy -o ast_expr.c  
make: bison: Command not found  
make: *** [ast_expr.c] Error 127
```

The following files are required in order to install Asterisk; they can be installed with the `yum install bison` command:

- *bison*
- *m4*

/usr/bin/ld: cannot find -lssl

The OpenSSL development packages are required by Asterisk within the *res_crypto.so* module for RSA key checks performed by the IAX2 protocol. If the OpenSSL development packages are not installed, the following error will occur:

```
/usr/bin/ld: cannot find -lssl
collect2: ld returned 1 exit status
make: *** [asterisk] Error 1
```

To install the OpenSSL development library, you'll require the following dependencies:

- *openssl-devel*
- *e2fsprogs-devel*
- *zlib-devel*
- *krb5-devel*
- *krb5-libs*

You can use the `yum install openssl-devel` command to install these files.

rpmbuild: command not found

To use the `make rpm` command, you must have the Red Hat Package Manager (RPM) development package installed. The following error will be encountered if it is absent:

```
make[1]: Leaving directory `/usr/src/asterisk-1.0.3'
/bin/sh: line 1: rpmbuild: command not found
make: *** [rpm] Error 127
```

You can install the build environment with `yum install rpmbuild`.

Zaptel

You may also run into errors when compiling Zaptel. Here are some of the most commonly occurring problems, and what to do about them.

make: cc: Command not found

You will receive the following error if you attempt to build Zaptel without the `gcc` compiler installed:

```
make: cc: Command not found
make: *** [gendigits.o] Error 127
```

Be sure to install *gcc* and its dependencies. For more information, see “C compiler cannot create executables” in the previous section.

FATAL: Module *wctdm*/*fxs*/*fxo* not found

The TDM400P cards require the PCI bus to be Version 2.2. If you attempt to load the Zapata telephony drivers with an older version, you may get the following errors:

- When attempting to load the *wctdm* driver, you may see this error:
FATAL: Module *wctdm* not found
- When attempting to load the *wctdm* or *wcfxo* driver, you may see an error such as this:
ZT_CHANCONFIG failed on channel 1: No such device or address (6)
FATAL: Module *wctdm* not found

The only way to resolve these errors is to use a newer motherboard that supports PCI Version 2.2.



You may also encounter these errors if the power has not been attached to the Molex connector found on the TDM400P card.

Unresolved symbol link when loading *ztdummy*

The *ztdummy* driver requires that a UHCI USB controller be available on Linux 2.4 kernels (the USB controller is not a requirement on Linux 2.6 kernels, because they are capable of generating the 1-kHz timing reference). There exists a secondary kind of controller, known as OHCI, which is not compatible with the *ztdummy* driver. If the UHCI USB controller is not accessible on Linux 2.4 kernels, the following error will occur:

```
/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved
symbol unlink_td
/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved
symbol alloc_td
/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved
symbol delete_desc
/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved
symbol uhci_devices
/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved
symbol uhci_interrupt
/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved
symbol fill_td
/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved
symbol insert_td_horizontal
/lib/modules/2.4.22/misc/ztdummy.o: insmod /lib/modules/2.4.22/misc/ztdummy.o failed
/lib/modules/2.4.22/misc/ztdummy.o: insmod ztdummy failed
```

You can verify that you have the correct style of USB controller and its associated drivers with the `lsmod` command:

```
# lsmod
Module                Size  Used by
usb_uhci               26412  0
usbcore               79040  1 [hid usb-uhci]
```

As you can see in the example above, you are looking to make sure that the `usbcore` and `usb_uhci` modules are loaded. If these modules are not loaded, be sure that USB has been activated within your BIOS and that the modules exist and are being loaded.

If the USB drivers are not loaded, you can still check which type of USB controller you have with the `dmesg` command:

```
# dmesg | grep -i usb
```

To verify that you indeed have a UHCI USB controller, look for the following lines:

```
uhci_hcd 0000:00:04.2: new USB bus registered, assigned bus number 1
hub 1-0:1.0: USB hub found
uhci_hcd 0000:00:04.3: new USB bus registered, assigned bus number 2
hub 2-0:1.0: USB hub found
```

Depmod errors during compilation

If you experience `depmod` errors during compilation, you more than likely don't have a symbolic link to your Linux kernel sources. If you don't have your Linux kernel sources installed, retrieve the sources for your installed kernel, install them, and create a symbolic link against `/usr/src/linux-2.4`. The following is an example of a `depmod` error:

```
depmod: *** Unresolved symbols in /lib/modules/2.4.22/kernel/drivers/block/loop.o
```

Loading Zaptel Modules

In this section, we'll take a quick look at how to load the `zaptel` and `ztdummy` modules. The `zaptel` module does not require any configuration if it's being used only for the `ztdummy` module. If you plan on loading the `ztdummy` module as your timing source (and thus, you will not be running any PCI hardware in your system), now is a good time to load both drivers.

Systems Running udevd

In the early days of Linux, the system's `/dev/` directory was populated with a list of devices with which the system could potentially interact. At the time, nearly 18,000 devices were listed. That all changed when `devfs` was released, allowing dynamic creation of devices that are active within the system. Some of the recently released

distributions have incorporated the *udev* daemon into their systems to dynamically populate */dev/* with device nodes.

To allow Zaptel and other device drivers to access the PCI hardware installed in your system, you must add some rules. Using your favorite text editor, open up your *udev*d rules file. On Fedora Core 3, for example, this file is located at */etc/udev/rules.d/50-udev.rules*. Add the following lines to the end of your rules file:

```
# Section for zaptel device
KERNEL="zapctl",    NAME="zap/ctl"
KERNEL="zaptimer",  NAME="zap/timer"
KERNEL="zapchannel", NAME="zap/channel"
KERNEL="zappseudo", NAME="zap/pseudo"
KERNEL="zap[0-9]*", NAME="zap/%n"
```

Save the file and reboot your system for the settings to take effect.

Loading Zaptel

The *zaptel* module must be loaded before any of the other modules are loaded and used. Note that if you will be using the *zaptel* module with PCI hardware, you must configure */etc/zaptel.conf* before you load it. (We will discuss how to configure *zaptel.conf* for use with hardware in Chapter 4.) If you are using *zaptel* only to access *ztdummy*, you can load it with the *modprobe* command, as follows:

```
# modprobe zaptel
```

If all goes well, you shouldn't see any output. To verify that the *zaptel* module loaded successfully, use the *lsmod* command. You should be returned a line showing the *zaptel* module and the amount of memory it is using:

```
# lsmod | grep zaptel
zaptel                201988  0
```

Loading ztdummy

The *ztdummy* module is an interface to a device that provides timing, which in turn allows Asterisk to provide timing to various applications and functions that require it. Use the *modprobe* command to load the *ztdummy* module after *zaptel* has been loaded:

```
# modprobe ztdummy
```

If *ztdummy* loads successfully, no output will be displayed. To verify that *ztdummy* is loaded and is being used by *zaptel*, use the *lsmod* command. The following output is from a computer running the 2.6 kernel:

```
# lsmod | grep ztdummy
Module                Size Used by
ztdummy                3796  0
zaptel                 201988 1 ztdummy
```

If you happen to be running a 2.4 kernel-based computer, your output from `lsmod` will show that `ztdummy` is using the `usb-uhci` module:

```
# lsmod | grep ztdummy
Module                Size  Used by
ztdummy                3796  0
zaptel                 201988 0 ztdummy
usb-uhci               24524 0 ztdummy
```

Loading libpri

The *libpri* libraries do not need to be loaded like modules. Asterisk looks for *libpri* at compile time and configures itself to use the libraries if they are found.

Loading Asterisk

Asterisk can be loaded in a variety of ways. The easiest way is to start Asterisk by running the binary file directly from the Linux command-line interface. If you are running a system that uses the *init.d* scripts, you can easily start and restart Asterisk that way as well. However, the preferred way of starting Asterisk is via the *safe_asterisk* script.

CLI Commands

The Asterisk binary is, by default, located at `/usr/sbin/asterisk`. If you run `/usr/sbin/asterisk`, it will be loaded as a daemon. There are also a few switches you should be aware of that allow you to (re)connect to the Asterisk CLI, set the verbosity of CLI output, and allow core dumps if Asterisk crashes (for debugging with *gdb*). To explore the full range of options, run Asterisk with the `-h` switch:

```
# /usr/sbin/asterisk -h
```

Here is a list of the most commonly used options:

- c
Console. This allows you to connect to the Asterisk CLI.
- v
Verbosity. This is used to set the amount of output for CLI debugging.
- g
Core dump. If Asterisk were to crash unexpectedly, this would cause a core file to be created for later tracing with *gdb*.
- r
Remote. This is used to reconnect remotely to an already running Asterisk process. (The process is remote from the standpoint of the console connecting to it)

but is actually a local process on the machine. This has nothing to do with connecting to a remote process over a network using a protocol such as IP, as this is not supported.)

```
-rx "restart now"
```

Execute. Using this command in combination with `-r` allows you to execute a CLI command without having to connect to the CLI and type it manually.

Let's look at some examples. To start Asterisk and connect to the CLI with a verbosity level of 3, use the following command:

```
# /usr/sbin/asterisk -cvvv
```

If the Asterisk process is already running (for example, if you started Asterisk with `/usr/sbin/asterisk`), instead use the reconnect switch, like so:

```
# /usr/sbin/asterisk -vvvr
```

If you want Asterisk to dump a core file after a crash, you can use the `-g` switch when starting Asterisk:

```
# /usr/sbin/asterisk -g
```

To execute a command without connecting to the CLI and typing it (perhaps for use within a script), you can use the `-x` switch in combination with the `-r` switch:

```
# /usr/sbin/asterisk -rx "restart now"
```

If you are experiencing crashes and would like to output to a debug file, use the following command:

```
# /usr/sbin/asterisk -vvvvvvvvc | tee /tmp/debug.log
```

Red Hat–Style Initialization Script

If you ran the `make config` command earlier (or manually copied the initialization scripts), you can start and restart Asterisk with the following commands:

```
# /etc/rc.d/init.d/asterisk start
# /etc/rc.d/init.d/asterisk stop
```

The `safe_asterisk` Script

The main purpose of the `safe_asterisk` script is to dump a core file if Asterisk fails and to automatically restart it. There is also a notify option within the script, which, if set, will send an email letting you know that Asterisk died unexpectedly. An added benefit of the script is that it will load the Asterisk CLI on terminal interface 9 (by default; this is configurable), so you can easily switch to that window to monitor your Asterisk system.

The default location of the *safe_asterisk* script is */usr/sbin/safe_asterisk*, and it can be executed as such. Let's review the various options contained in the *safe_asterisk* script:

```
CLIARGS="$*"          # Grab any args passed to safe_asterisk
TTY=9                 # TTY (if you want one) for Asterisk to run on
CONSOLE=yes           # Whether or not you want a console
#NOTIFY=ben@alkaloid.net # Email address for crash notifications
```

The first line simply allows you to pass arguments to the *safe_asterisk* script from the Linux CLI; it should not be edited directly. *TTY=9* specifies the Linux console on which to run the Asterisk CLI output. You can disable this feature by specifying *CONSOLE=no*. If you would like to be notified if Asterisk dies suddenly and requires a restart, uncomment the *NOTIFY* line and replace *ben@alkaloid.net* with your email address. Note that the crash notifications are sent with the *mail* command, so your system must be set up to process and send email.

Directories Used by Asterisk

Asterisk uses several directories on a Linux system to manage the various aspects of the system, such as voicemail recordings, voice prompts, and configuration files. This section discusses the necessary directories, all of which are created during installation and configured in the *asterisk.conf* file.

/etc/asterisk/

The */etc/asterisk/* directory contains the Asterisk configuration files. One file, however—*zaptel.conf*—is located in the */etc/* directory. The Zaptel hardware was originally designed by Jim Dixon of the Zapata Telephony Group as a way of bringing reasonable and affordable computer telephony equipment to the world. Asterisk makes use of this hardware, but any other software can also make use of the Zaptel hardware and drivers. Consequently, the *zaptel.conf* configuration file is not directly located in the */etc/asterisk/* directory.

/usr/lib/asterisk/modules/

The */usr/lib/asterisk/modules/* directory contains all the Asterisk loadable modules. Within this directory are the various applications, codecs, formats, and channels used by Asterisk. By default, Asterisk loads all of these modules at startup. You can disable any modules you are not using in the *modules.conf* file, but be aware that certain modules are required by Asterisk or are dependencies of other modules. Attempting to load Asterisk without these modules will cause an error at startup.

/var/lib/asterisk

The */var/lib/asterisk/* directory contains the *astdb* file and a number of subdirectories. The *astdb* file contains the local Asterisk database information, which is somewhat like the Microsoft Windows Registry. The Asterisk database is a simple implementation based on v1 of the Berkeley database. The *db.c* file in the Asterisk source states that this version was chosen for the following reason: “DB3 implementation is released under an alternative license incompatible with the GPL. Thus in order to keep Asterisk licensing simplistic, it was decided to use version 1 as it is released under the BSD license.”

The subdirectories within */var/lib/asterisk/* include:

agi-bin/

The *agi-bin/* directory contains your custom scripts, which can interface with Asterisk via the various built-in AGI applications. For more information about AGI, see Chapter 8.

firmware/

The *firmware/* directory contains firmware for various Asterisk-compatible devices. It currently contains only the *iax/* subdirectory, which holds the binary firmware image for Digium’s IAXy.

images/

Applications that communicate with channels supporting graphical images look in the *images/* directory. Most channels do not support the transmission of images, so this directory is rarely used. However, if more devices that support and make use of graphical images are released, this directory will become more relevant.

keys/

Asterisk can use a public/private key system to authenticate peers connecting to your box via an RSA digital signature. If you place a peer’s public key in your *keys/* directory, that peer can be authenticated by channels supporting this method (such as the IAX2 channels). The private key is never distributed to the public. The reverse is also true: you can distribute your public key to your peers, allowing you to be authenticated with the use of your private key. Both the public and private keys—ending in the *.pub* and *.key* file extensions, respectively—are stored in the *keys/* directory.

mohmp3/

When you configure Asterisk for Music on Hold, applications utilizing this feature look for their MP3 files in the *mohmp3/* directory. Asterisk is a bit picky about how the MP3 files are formatted, so you should use constant bitrate (CBR) encoding and strip the ID3 tags from your files.

sounds/

All of the available voice prompts for Asterisk reside in the *sounds/* directory. The contents of the basic prompts included with Asterisk are in the *sounds.txt* file located in your Asterisk source code directory. Contents of the additional prompts are located in the *sounds-extra.txt* file in the directory to which you extracted the *asterisk-sounds* package earlier in this chapter.

/var/spool/asterisk/

The Asterisk spool directory contains several subdirectories, including *outgoing/*, *qcall/*, *tmp/*, and *voicemail/* (see Figure 3-2). Asterisk monitors the *outgoing* and *qcall* directories for text files containing call request information. These files allow you to generate a call simply by copying or moving the correctly structured file into the *outgoing/* directory.

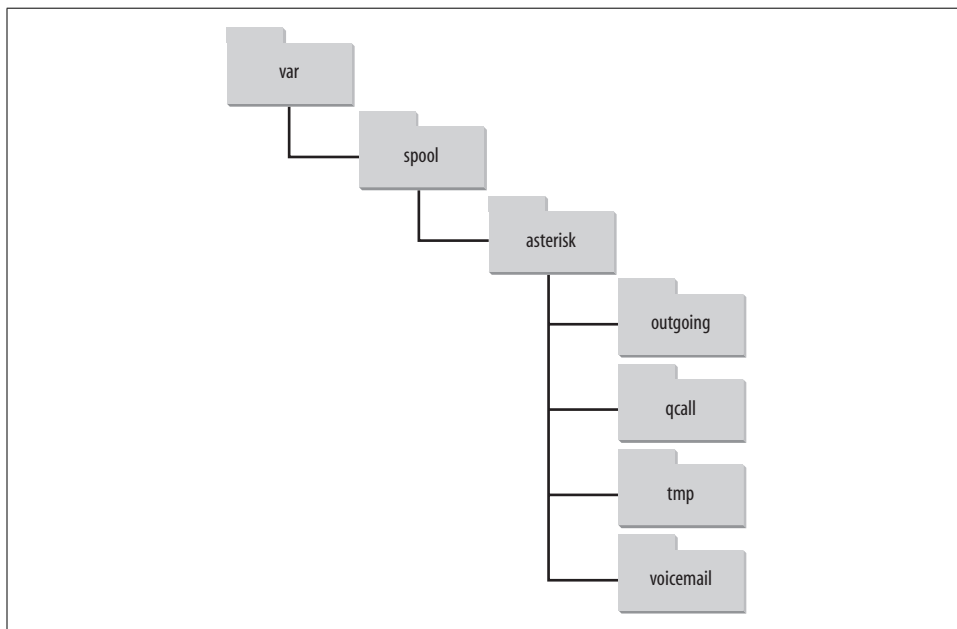


Figure 3-2. */var/spool/asterisk/* directory structure

The old (now deprecated) *qcall* method of generating calls utilized a single line of text within the call file. Call files for use within the *qcall* directory took the form of:

```
Dialstring Caller-ID Extension Maxsecs [Identifier] [Required-response]
```

This rather limited what you could do with the call file, and what kinds of information you could pass to Asterisk. Thus, a new spooling method was developed in Asterisk, using the *outgoing* directory. Call files being placed into this directory can

contain much more valuable information, such as the Context, Extension, and Priority where the answered call should start, or simply the application and its arguments. You can also set variables and specify an account code for Call Detail Records. More information about the use of call files is presented in Chapter 9.

The *tmp/* directory is used, funny enough, to hold temporary information. Certain applications may require a place to write files to before copying the complete files to their final destinations. This prevents two processes from trying to write to and read from a file at the same time.

All voicemail and user greetings are contained within the *voicemail/* directory. Extensions configured in *voicemail.conf* that have been logged into at least once are created as subdirectories of *voicemail/*.

/var/run/

The */var/run/* directory contains the process ID (pid) information for all active processes on the system, including Asterisk (as specified in the *asterisk.conf* file). Note that */var/run/* is OS-dependent and may differ.

/var/log/asterisk/

The */var/log/asterisk/* directory is where Asterisk logs information. You can control the type of information being logged to the various files by editing the *logger.conf* file located in the */etc/asterisk/* directory. Basic configuration of the *logger.conf* file is covered in Appendix E.

/var/log/asterisk/cdr-csv

The */var/log/asterisk/cdr-csv* directory is used to store the CDRs in comma-separated value (CSV) format. By default information is stored in the *Master.csv* file, but individual accounts can store their own CDRs in separate files with the use of the *accountcode* option (see Appendix A for more information).

Conclusion

In this chapter, we have reviewed the procedures for obtaining, compiling, and installing Asterisk and the associated packages. In the following chapter, we will touch on the initial configuration of your system with regard to various communications channels, such as analog devices attached to FXS and FXO ports, SIP channels, and IAX2 endpoints.