Bacula Installation and Configuration Guide

It comes in the night and sucks the essence from your computers.

Kern Sibbald

December 8, 2009
This manual documents Bacula version 3.0.3 (18 October 2009)

Contents

Chapter 1

Getting Started with Bacula

If you are like me, you want to get Bacula running immediately to get a feel for it, then later you want to go back and read about all the details. This chapter attempts to accomplish just that: get you going quickly without all the details. If you want to skip the section on Pools, Volumes and Labels, you can always come back to it, but please read to the end of this chapter, and in particular follow the instructions for testing your tape drive.

We assume that you have managed to build and install Bacula, if not, you might want to first look at the System Requirements then at the Compiling and Installing Bacula chapter of this manual.

1.1 Understanding Jobs and Schedules

In order to make Bacula as flexible as possible, the directions given to Bacula are specified in several pieces. The main instruction is the job resource, which defines a job. A backup job generally consists of a FileSet, a Client, a Schedule for one or several levels or times of backups, a Pool, as well as additional instructions. Another way of looking at it is the FileSet is what to backup; the Client is who to backup; the Schedule defines when, and the Pool defines where (i.e. what Volume).

Typically one FileSet/Client combination will have one corresponding job. Most of the directives, such as FileSets, Pools, Schedules, can be mixed and matched among the jobs. So you might have two different Job definitions (resources) backing up different servers using the same Schedule, the same Fileset (backing up the same directories on two machines) and maybe even the same Pools. The Schedule will define what type of backup will run when (e.g. Full on Monday, incremental the rest of the week), and when more than one job uses the same schedule, the job priority determines which actually runs first. If you have a lot of jobs, you might want to use JobDefs, where you can set defaults for the jobs, which can then be changed in the job resource, but this saves rewriting the identical parameters for each job. In addition to the FileSets you want to back up, you should also have a job that backs up your catalog.

Finally, be aware that in addition to the backup jobs there are restore, verify, and admin jobs, which have different requirements.

1.2 Understanding Pools, Volumes and Labels

If you have been using a program such as **tar** to backup your system, Pools, Volumes, and labeling may be a bit confusing at first. A Volume is a single physical tape (or possibly a single file) on which Bacula will write your backup data. Pools group together Volumes so that a backup is not restricted to the length of a single Volume (tape). Consequently, rather than explicitly naming Volumes in your Job, you specify a Pool, and Bacula will select the next appendable Volume from the Pool and request you to mount it.

Although the basic Pool options are specified in the Director's Pool resource, the **real** Pool is maintained in the Bacula Catalog. It contains information taken from the Pool resource (bacula-dir.conf) as well as information on all the Volumes that have been added to the Pool. Adding Volumes to a Pool is usually done manually with the Console program using the **label** command.

For each Volume, Bacula maintains a fair amount of catalog information such as the first write date/time, the last write date/time, the number of files on the Volume, the number of bytes on the Volume, the number of Mounts, etc.

Before Bacula will read or write a Volume, the physical Volume must have a Bacula software label so that Bacula can be sure the correct Volume is mounted. This is usually done using the **label** command in the Console program.

The steps for creating a Pool, adding Volumes to it, and writing software labels to the Volumes, may seem tedious at first, but in fact, they are quite simple to do, and they allow you to use multiple Volumes (rather than being limited to the size of a single tape). Pools also give you significant flexibility in your backup process. For example, you can have a "Daily" Pool of Volumes for Incremental backups and a "Weekly" Pool of Volumes for Full backups. By specifying the appropriate Pool in the daily and weekly backup Jobs, you thereby insure that no daily Job ever writes to a Volume in the Weekly Pool and vice versa, and Bacula will tell you what tape is needed and when.

For more on Pools, see the Pool Resource section of the Director Configuration chapter, or simply read on, and we will come back to this subject later.

1.3 Setting Up Bacula Configuration Files

After running the appropriate **./configure** command and doing a **make**, and a **make install**, if this is the first time you are running Bacula, you must create valid configuration files for the Director, the File daemon, the Storage daemon, and the Console programs. If you have followed our recommendations, default configuration files as well as the daemon binaries will be located in your installation directory. In any case, the binaries are found in the directory you specified on the **--sbindir** option to the **./configure** command, and the configuration files are found in the directory you specified on the **--sysconfdir** option.

When initially setting up Bacula you will need to invest a bit of time in modifying the default configuration files to suit your environment. This may entail starting and stopping Bacula a number of times until you get everything right. Please do not despair. Once you have created your configuration files, you will rarely need to change them nor will you stop and start Bacula very often. Most of the work will simply be in changing the tape when it is full.

1.3.1 Configuring the Console Program

The Console program is used by the administrator to interact with the Director and to manually start/stop Jobs or to obtain Job status information.

The Console configuration file is found in the directory specified on the **--sysconfdir** option that you specified on the **./configure** command and by default is named **bconsole.conf**.

If you choose to build the GNOME console with the **--enable-gnome** option, you also find a default configuration file for it, named **bgnome-console.conf**.

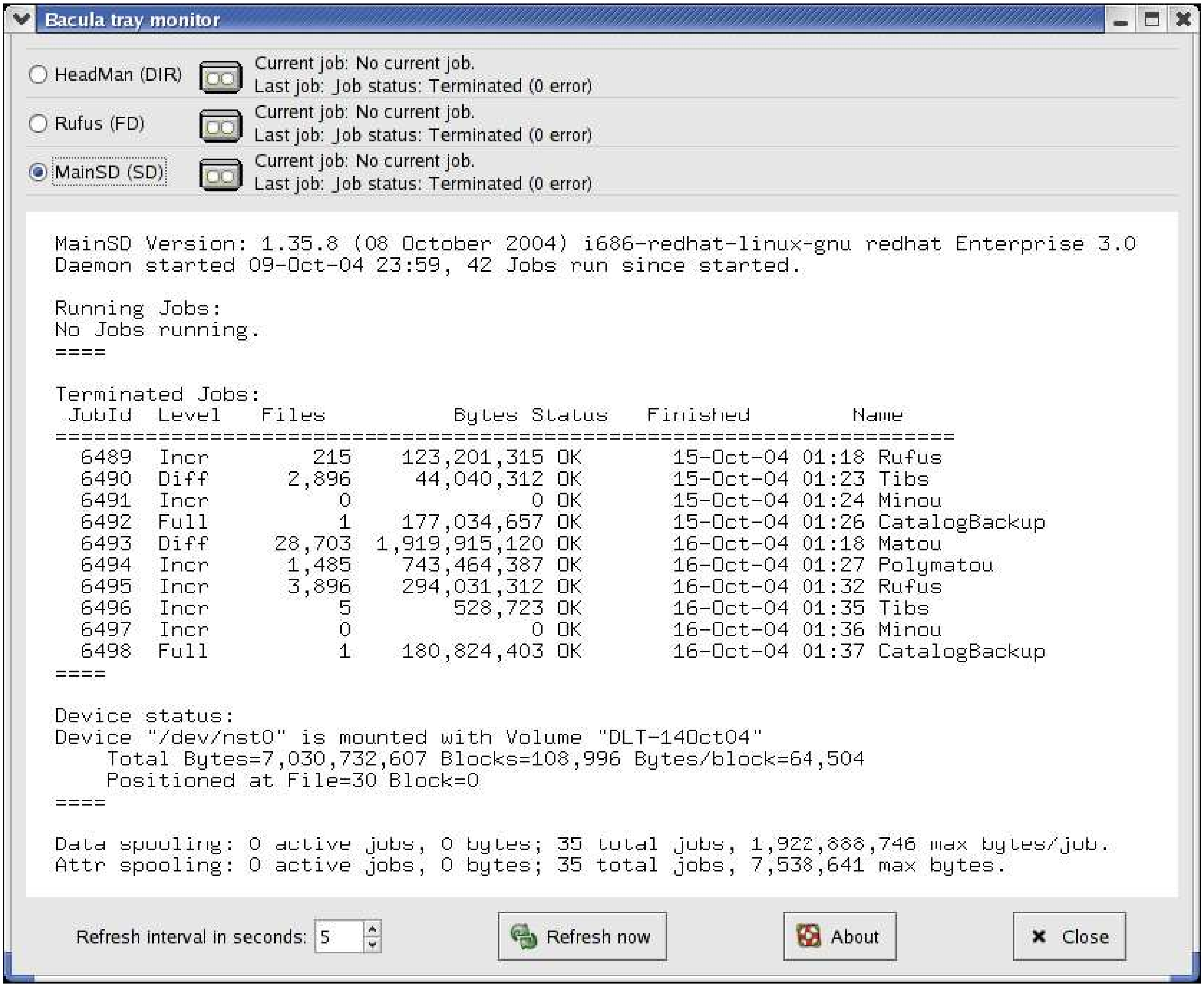
The same applies to the wxWidgets console, which is build with the **--enable-bwx-console** option, and the name of the default configuration file is, in this case, **bwx-console.conf**.

Normally, for first time users, no change is needed to these files. Reasonable defaults are set.

Further details are in the Console configuration chapter.

1.3.2 Configuring the Monitor Program

The Monitor program is typically an icon in the system tray. However, once the icon is expanded into a full window, the administrator or user can obtain status information about the Director or the backup status on the local workstation or any other Bacula daemon that is configured.



The image shows a tray-monitor configured for three daemons. By clicking on the radio buttons in the upper left corner of the image, you can see the status for each of the daemons. The image shows the status for the Storage daemon (MainSD) that is currently selected.

The Monitor configuration file is found in the directory specified on the **--sysconfdir** option that you specified on the **./configure** command and by default is named **tray-monitor.conf**. Normally, for first time users, you just need to change the permission of this file to allow non-root users to run the Monitor, as this application must run as the same user as the graphical environment (don't forget to allow non-root users to execute **bacula-tray-monitor**). This is not a security problem as long as you use the default settings.

More information is in the Monitor configuration chapter.

1.3.3 Configuring the File daemon

The File daemon is a program that runs on each (Client) machine. At the request of the Director, finds the files to be backed up and sends them (their data) to the Storage daemon.

The File daemon configuration file is found in the directory specified on the **--sysconfdir** option that you specified on the **./configure** command. By default, the File daemon's configuration file is named **bacula-fd.conf**. Normally, for first time users, no change is needed to this file. Reasonable defaults are set. However, if you are going to back up more than one machine, you will need to install the File daemon with a unique configuration file on each machine to be backed up. The information about each File daemon must appear in the Director's configuration file.

Further details are in the File daemon configuration chapter.

1.3.4 Configuring the Director

The Director is the central control program for all the other daemons. It schedules and monitors all jobs to be backed up.

The Director configuration file is found in the directory specified on the **--sysconfdir** option that you specified on the **./configure** command. Normally the Director's configuration file is named **bacula-dir.conf**.

In general, the only change you must make is modify the FileSet resource so that the **Include** configuration directive contains at least one line with a valid name of a directory (or file) to be saved.

If you do not have a DLT tape drive, you will probably want to edit the Storage resource to contain names that are more representative of your actual storage device. You can always use the existing names as you are free to arbitrarily assign them, but they must agree with the corresponding names in the Storage daemon's configuration file.

You may also want to change the email address for notification from the default **root** to your email address.

Finally, if you have multiple systems to be backed up, you will need a separate File daemon or Client specification for each system, specifying its name, address, and password. We have found that giving your daemons the same name as your system but post fixed with **-fd** helps a lot in debugging. That is, if your system name is **foobaz**, you would give the File daemon the name **foobaz-fd**. For the Director, you should use **foobaz-dir**, and for the storage daemon, you might use **foobaz-sd**. Each of your Bacula components **must** have a unique name. If you make them all the same, aside from the fact that you will not know what daemon is sending what message, if they share the same working directory, the daemons temporary file names will not be unique, and you will get many strange failures.

More information is in the Director configuration chapter.

1.3.5 Configuring the Storage daemon

The Storage daemon is responsible, at the Director's request, for accepting data from a File daemon and placing it on Storage media, or in the case of a restore request, to find the data and send it to the File daemon.

The Storage daemon's configuration file is found in the directory specified on the **--sysconfdir** option that you specified on the **./configure** command. By default, the Storage daemon's file is named **bacula-sd.conf**. Edit this file to contain the correct Archive device names for any tape devices that you have. If the configuration process properly detected your system, they will already be correctly set. These Storage resource name and Media Type must be the same as the corresponding ones in the Director's configuration file **bacula-dir.conf**. If you want to backup to a file instead of a tape, the Archive device must point to a directory in which the Volumes will be created as files when you label the Volume.

Further information is in the Storage daemon configuration chapter.

1.4 Testing your Configuration Files

You can test if your configuration file is syntactically correct by running the appropriate daemon with the **-t** option. The daemon will process the configuration file and print any error messages then terminate. For example, assuming you have installed your binaries and configuration files in the same directory.

```
cd <installation-directory>
./bacula-dir -t -c bacula-dir.conf
./bacula-fd -t -c bacula-fd.conf
./bacula-sd -t -c bacula-sd.conf
./bconsole -t -c bconsole.conf
./bgnome-console -t -c bgnome-console.conf
./bwx-console -t -c bwx-console.conf
./bat -t -c bat.conf
su <normal user> -c "./bacula-tray-monitor -t -c tray-monitor.conf"
```

will test the configuration files of each of the main programs. If the configuration file is OK, the program will terminate without printing anything. Please note that, depending on the configure options you choose, some, or even all, of the three last commands will not be available on your system. If you have installed the binaries in traditional Unix locations rather than a single file, you will need to modify the above commands appropriately (no ./ in front of the command name, and a path in front of the conf file name).

1.5 Testing Compatibility with Your Tape Drive

Before spending a lot of time on Bacula only to find that it doesn't work with your tape drive, please read the **Testing Your Tape Drive** chapter of this manual. If you have a modern standard SCSI tape drive on a Linux or Solaris, most likely it will work, but better test than be sorry. For FreeBSD (and probably other xBSD flavors), reading the above mentioned tape testing chapter is a must. Also, for FreeBSD, please see The FreeBSD Diary for a detailed description on how to make Bacula work on your system. In addition, users of FreeBSD prior to 4.9-STABLE dated Mon Dec 29 15:18:01 2003 UTC who plan to use tape devices, please see the file **platforms/freebsd/pthreads-fix.txt** in the main Bacula directory concerning important information concerning compatibility of Bacula and your system.

1.6 Get Rid of the /lib/tls Directory

The new pthreads library **/lib/tls** installed by default on recent Red Hat systems running Linux kernel 2.4.x is defective. You must remove it or rename it, then reboot your system before running Bacula otherwise after a week or so of running, Bacula will either block for long periods or deadlock entirely. You may want to use the loader environment variable override rather than removing /lib/tls. Please see Supported Operating Systems for more information on this problem.

This problem does not occur on systems running Linux 2.6.x kernels.

1.7 Running Bacula

Probably the most important part of running Bacula is being able to restore files. If you haven't tried recovering files at least once, when you actually have to do it, you will be under a lot more pressure, and prone to make errors, than if you had already tried it once.

To get a good idea how to use Bacula in a short time, we **strongly** recommend that you follow the example in the Running Bacula Chapter of this manual where you will get detailed instructions on how to run Bacula.

1.8 Log Rotation

If you use the default **bacula-dir.conf** or some variation of it, you will note that it logs all the Bacula output to a file. To avoid that this file grows without limit, we recommend that you copy the file **logrotate** from the **scripts/logrotate** to **/etc/logrotate.d/bacula**. This will cause the log file to be rotated once a month and kept for a maximum of five months. You may want to edit this file to change the default log rotation preferences.

1.9 Log Watch

Some systems such as Red Hat and Fedora run the logwatch program every night, which does an analysis of your log file and sends an email report. If you wish to include the output from your Bacula jobs in that report, please look in the **scripts/logwatch** directory. The **README** file in that directory gives a brief explanation on how to install it and what kind of output to expect.

1.10 Disaster Recovery

If you intend to use Bacula as a disaster recovery tool rather than simply a program to restore lost or damaged files, you will want to read the Disaster Recovery Using Bacula Chapter of this manual.

In any case, you are strongly urged to carefully test restoring some files that you have saved rather than wait until disaster strikes. This way, you will be prepared.

Chapter 2

Installing Bacula

In general, you will need the Bacula source release, and if you want to run a Windows client, you will need the Bacula Windows binary release. However, Bacula needs certain third party packages (such as **MySQL**, **PostgreSQL**, or **SQLite** to build and run properly depending on the options you specify. Normally, **MySQL** and **PostgreSQL** are packages that can be installed on your distribution. However, if you do not have them, to simplify your task, we have combined a number of these packages into three **depkgs** releases (Dependency Packages). This can vastly simplify your life by providing you with all the necessary packages rather than requiring you to find them on the Web, load them, and install them.

2.1 Source Release Files

Beginning with Bacula 1.38.0, the source code has been broken into four separate tar files each corresponding to a different module in the Bacula SVN. The released files are:

bacula-2.2.8.tar.gz This is the primary source code release for Bacula. On each release the version number (2.2.8) will be updated.

bacula-docs-2.2.8.tar.gz This file contains a copy of the docs directory with the documents prebuild. English HTML directory, single HTML file, and pdf file. The French and German translations are in progress, but are not built.

bacula-gui-2.2.8.tar.gz This file contains the non-core GUI programs. Currently, it contains bacula-web, a PHP program for producing management viewing of your Bacula job status in a browser; and bimagemgr a browser program for burning CDROM images with Bacula Volumes.

bacula-rescue-2.0.0.tar.gz This is the Bacula Rescue CDROM code. Note, the version number of this package is not tied to the Bacula release version, so it will be different. Using this code, you can burn a CDROM with your system configuration and containing a statically linked version of the File daemon. This can permit you to easily repartition and reformat your hard disks and reload your system with Bacula in the case of a hard disk failure. Unfortunately this rescue disk does not properly boot for all Linux distributions. The problem is that the boot procedure can vary significantly between distributions, and even within a distribution, they are a moving target.

This package evolves slower than the Bacula source code, so there may not always be a new release of the rescue package when making minor updates to the Bacula code. For example, when releasing Bacula version 2.2.8, the rescue package may still be at version 2.0.0 if there were no updates.

winbacula-2.2.8.exe This file is the 32 bit Windows installer for installing the Windows client (File daemon) on a Windows machine. This client will also run on 64 bit Windows machines. Beginning with Bacula version 1.39.20, this executable will also optionally load the Win32 Director and the Win32 Storage daemon.

2.2 Upgrading Bacula

If you are upgrading from one Bacula version to another, you should first carefully read the ReleaseNotes of all major versions between your current version and the version to which you are upgrading. In many upgrades, especially for minor patch upgrades (e.g. between 3.0.0 and 3.0.1) there will be no database upgrade, and hence the process is rather simple.

With version 3.0.0 and later, you **must** ensure that on any one machine that all components of Bacula are running on exactly the same version. Prior to version 3.0.0, it was possible to run a lower level FD with a newer Director and SD. This is no longer the case.

As always, we attempt to support older File daemons. This avoids the need to do a simultaneous upgrade of many machines. For exactly what older versions of the FD are supported, please see the ReleaseNotes for the new version. In any case, you must always upgrade both the Director and the Storage daemon at the same time, and you must also upgrade any File daemon that is running on the same machine as a Director or a Storage daemon (see the prior paragraph).

If the Bacula catalog database has been upgraded (as it is almost every major release), you will either need to reinitialize your database starting from scratch (not normally a good idea), or save an ASCII copy of your database, then proceed to upgrade it. If you are upgrading two major versions (e.g. 1.36 to 2.0) then life will be more complicated because you must do two database upgrades. See below for more on this.

Upgrading the catalog is normally done after Bacula is build and installed by:

```
cd <installed-scripts-dir> (default /etc/bacula)
./update_bacula_tables
```

This update script can also be find in the Bacula source src/cats directory.

If there are several database upgrades between your version and the version to which you are upgrading, you will need to apply each database upgrade script. For your convenience, you can find all the old upgrade scripts in the **upgradedb** directory of the source code. You will need to edit the scripts to correspond to your system configuration. The final upgrade script, if any, can be applied as noted above.

If you are upgrading from one major version to another, you will need to replace all your components at the same time as generally the inter-daemon protocol will change. However, within any particular release (e.g. version 1.32.x) unless there is an oversight or bug, the daemon protocol will not change. If this is confusing, simply read the ReleaseNotes very carefully as they will note if all daemons must be upgraded at the same time.

Finally, please note that in general it is not necessary or desirable to do a **make uninstall** before doing an upgrade providing you are careful not to change the installation directories. In fact, if you do so, you will most likely delete all your conf files, which could be disastrous. The normal procedure during an upgrade is simply:

```
./configure (your options)
make
make install
```

In general none of your existing .conf or .sql files will be overwritten, and you must do both the **make** and **make install** commands, a **make install** without the preceding **make** will not work.

For additional information on upgrading, please see the Upgrading Bacula Versions in the Tips chapter of this manual.

2.3 Releases Numbering

Every Bacula release whether beta or production has a different number as well as the date of the release build. The numbering system follows traditional Open Source conventions in that it is of the form.

```
major.minor.release
```

For example:

```
1.38.11
```

where each component (major, minor, patch) is a number. The major number is currently 1 and normally does not change very frequently. The minor number starts at 0 and increases each for each production release by 2 (i.e. it is always an even number for a production release), and the patch number is starts at zero each time the minor number changes. The patch number is increased each time a bug fix (or fixes) is released to production.

So, as of this date (10 September 2006), the current production Bacula release is version 1.38.11. If there are bug fixes, the next release will be 1.38.12 (i.e. the patch number has increased by one).

For all patch releases where the minor version number does not change, the database and all the daemons will be compatible. That means that you can safely run a 1.38.0 Director with a 1.38.11 Client. Of course, in this case, the Director may have bugs that are not fixed. Generally, within a minor release (some minor releases are not so minor), all patch numbers are officially released to production. This means that while the current Bacula version is 1.38.11, versions 1.38.0, 1.38.1, ... 1.38.10 have all been previously released.

When the minor number is odd, it indicates that the package is under development and thus may not be stable. For example, while the current production release of Bacula is currently 1.38.11, the current development version is 1.39.22. All patch versions of the development code are available in the SVN (source repository). However, not all patch versions of the development code (odd minor version) are officially released. When they are released, they are released as beta versions (see below for a definition of what beta means for Bacula releases).

In general when the minor number increases from one production release to the next (i.e. 1.38.x to 1.40.0), the catalog database must be upgraded, the Director and Storage daemon must always be on the same minor release number, and often (not always), the Clients must also be on the same minor release. As often as possible, we attempt to make new releases that are downwards compatible with prior clients, but this is not always possible. You must check the release notes. In general, you will have fewer problems if you always run all the components on the same minor version number (i.e. all either 1.38.x or 1.40.x but not mixed).

Beta Releases

Towards the end of the development cycle, which typically runs one year from a major release to another, there will be several beta releases of the development code prior to a production release. As noted above, beta versions always have odd minor version numbers (e.g 1.37.x or 1.39.x). The purpose of the beta releases is to allow early adopter users to test the new code. Beta releases are made with the following considerations:

- The code passes the regression testing on FreeBSD, Linux, and Solaris machines.
- There are no known major bugs, or on the rare occasion that there are, they will be documented or already in the bugs database.
- Some of the new code/features may not yet be tested.
- Bugs are expected to be found, especially in the new code before the final production release.
- The code will have been run in production in at least one small site (mine).

- The Win32 client will have been run in production at least one night at that small site.
- The documentation in the manual is unlikely to be complete especially for the new features, and the Release Notes may not be fully organized.
- Beta code is not generally recommended for everyone, but rather for early adopters.

2.4 Dependency Packages

As discussed above, we have combined a number of third party packages that Bacula might need into the **depkgs** release. You can, of course, get the latest packages from the original authors or from your operating system supplier. The locations of where we obtained the packages are in the README file in each package. However, be aware that the packages in the depkgs files have been tested by us for compatibility with Bacula.

Typically, a dependency package will be named **depkgs-ddMMMyy.tar.gz** where **dd** is the day we release it, **MMM** is the abbreviated month (e.g. Jan), and **yy** is the year. An actual example is: **depkgs-07Apr02.tar.gz**. To install and build this package (if needed), you do the following:

1. Create a **bacula** directory, into which you will place both the Bacula source as well as the dependency package.
2. Detar the **depkgs** into the **bacula** directory.
3. cd bacula/depkgs
4. make

Although the exact composition of the dependency packages may change from time to time, the current makeup is the following:

3rd Party Package	depkgs	depkgs-qt
SQLite	X	
SQLite3	X	
mtx	X	
qt4		X
qwt		X

Note, some of these packages are quite large, so that building them can be a bit time consuming. The above instructions will build all the packages contained in the directory. However, when building Bacula, it will take only those pieces that it actually needs.

Alternatively, you can make just the packages that are needed. For example,

```
cd bacula/depkgs
make sqlite
```

will configure and build only the SQLite package.

You should build the packages that you will require in **depkgs** a prior to configuring and building Bacula, since Bacula will need them during the build process.

For more information on the **depkgs-qt** package, please read the INSTALL file in the main directory of that package. If you are going to build Qt4 using **depkgs-qt**, you must source the **qt4-paths** file included in the package prior to building Bacula. Please read the INSTALL file for more details.

Even if you do not use SQLite, you might find it worthwhile to build **mtx** because the **tapeinfo** program that comes with it can often provide you with valuable information about your SCSI tape drive (e.g. compression, min/max block sizes, ...). Note, most distros provide **mtx** as part of their release.

The **depkgs1** package is depreciated and previously contained readline, which should be available on all operating systems.

The **depkgs-win32** package is deprecated and no longer used in Bacula version 1.39.x and later. It was previously used to build the native Win32 client program, but this program is now built on Linux systems using cross-compiling. All the tools and third party libraries are automatically downloaded by executing the appropriate scripts. See src/win32/README.mingw32 for more details.

2.5 Supported Operating Systems

Please see the Supported Operating Systems section of the QuickStart chapter of this manual.

2.6 Building Bacula from Source

The basic installation is rather simple.

1. Install and build any **depkgs** as noted above. This should be unnecessary on most modern Operating Systems.

2. Configure and install MySQL or PostgreSQL (if desired). Installing and Configuring MySQL Phase I or Installing and Configuring PostgreSQL Phase I. If you are installing from rpms, and are using MySQL, please be sure to install **mysql-devel**, so that the MySQL header files are available while compiling Bacula. In addition, the MySQL client library **mysqlclient** requires the gzip compression library **libz.a** or **libz.so**. If you are using rpm packages, these libraries are in the **libz-devel** package. On Debian systems, you will need to load the **zlib1g-dev** package. If you are not using rpms or debs, you will need to find the appropriate package for your system.

   Note, if you already have a running MySQL or PostgreSQL on your system, you can skip this phase provided that you have built the thread safe libraries. And you have already installed the additional rpms noted above.

   SQLite is not supported on Solaris. This is because it frequently fails with bus errors. However SQLite3 may work.

3. Detar the Bacula source code preferably into the **bacula** directory discussed above.

4. **cd** to the directory containing the source code.

5. ./configure (with appropriate options as described below). Any path names you specify as options on the ./configure command line must be absolute paths and not relative.

6. Check the output of ./configure very carefully, especially the Install binaries and Install config directories. If they are not correct, please rerun ./configure until they are. The output from ./configure is stored in **config.out** and can be re-displayed at any time without rerunning the ./configure by doing **cat config.out**.

7. If after running ./configure once, you decide to change options and re-run it, that is perfectly fine, but before re-running it, you should run:

   ```
   make distclean
   ```

   so that you are sure to start from scratch and not have a mixture of the two options. This is because ./configure caches much of the information. The **make distclean** is also critical if you move the source directory from one machine to another. If the **make distclean** fails, just ignore it and continue on.

8. make If you get errors while linking in the Storage daemon directory (src/stored), it is probably because you have not loaded the static libraries on your system. I noticed this problem on a Solaris system. To correct it, make sure that you have not added **--enable-static-tools** to the **./configure** command.

   If you skip this step (**make**) and proceed immediately to the **make install** you are making two serious errors: 1. your install will fail because Bacula requires a **make** before a **make install**. 2. you are depriving yourself of the chance to make sure there are no errors before beginning to write files to your system directories.

9. make install Please be sure you have done a **make** before entering this command, and that everything has properly compiled and linked without errors.

10. If you are new to Bacula, we **strongly** recommend that you skip the next step and use the default configuration files, then run the example program in the next chapter, then come back and modify your configuration files to suit your particular needs.

11. Customize the configuration files for each of the three daemons (Directory, File, Storage) and for the Console program. For the details of how to do this, please see Setting Up Bacula Configuration Files in the Configuration chapter of this manual. We recommend that you start by modifying the default configuration files supplied, making the minimum changes necessary. Complete customization can be done after you have Bacula up and running. Please take care when modifying passwords, which were randomly generated, and the **Name**s as the passwords and names must agree between the configuration files for security reasons.

12. Create the Bacula MySQL database and tables (if using MySQL) Installing and Configuring MySQL Phase II or create the Bacula PostgreSQL database and tables Configuring PostgreSQL II or alternatively if you are using SQLite Installing and Configuring SQLite Phase II.

13. Start Bacula (**./bacula start**) Note. the next chapter shows you how to do this in detail.

14. Interface with Bacula using the Console program

15. For the previous two items, please follow the instructions in the Running Bacula chapter of this manual, where you will run a simple backup and do a restore. Do this before you make heavy modifications to the configuration files so that you are sure that Bacula works and are familiar with it. After that changing the conf files will be easier.

16. If after installing Bacula, you decide to "move it", that is to install it in a different set of directories, proceed as follows:

    ```
    make uninstall
    make distclean
    ./configure (your-new-options)
    make
    make install
    ```

If all goes well, the **./configure** will correctly determine which operating system you are running and configure the source code appropriately. Currently, FreeBSD, Linux (Red Hat), and Solaris are supported. The Bacula client (File daemon) is reported to work with MacOS X 10.3 is if readline support is not enabled (default) when building the client.

If you install Bacula on more than one system, and they are identical, you can simply transfer the source tree to that other system and do a "make install". However, if there are differences in the libraries or OS versions, or you wish to install on a different OS, you should start from the original compress tar file. If you do transfer the source tree, and you have previously done a ./configure command, you MUST do:

```
make distclean
```

prior to doing your new ./configure. This is because the GNU autoconf tools cache the configuration, and if you re-use a configuration for a Linux machine on a Solaris, you can be sure your build will fail. To avoid this, as mentioned above, either start from the tar file, or do a "make distclean".

In general, you will probably want to supply a more complicated **configure** statement to ensure that the modules you want are built and that everything is placed into the correct directories.

For example, on Fedora, Red Hat, or SuSE one could use the following:

```
CFLAGS="-g -Wall" \
  ./configure \
    --sbindir=$HOME/bacula/bin \
    --sysconfdir=$HOME/bacula/bin \
    --with-pid-dir=$HOME/bacula/bin/working \
    --with-subsys-dir=$HOME/bacula/bin/working \
    --with-mysql \
    --with-working-dir=$HOME/bacula/bin/working \
    --with-dump-email=$USER
```

The advantage of using the above configuration to start is that everything will be put into a single directory, which you can later delete once you have run the examples in the next chapter and learned how Bacula works. In addition, the above can be installed and run as non-root.

For the developer's convenience, I have added a **defaultconfig** script to the **examples** directory. This script contains the statements that you would normally use, and each developer/user may modify them to suit his needs. You should find additional useful examples in this directory as well.

The **--enable-conio** or **--enable-readline** options are useful because they provide a command line history and editing capability for the Console program. If you have included either option in the build, either the **termcap** or the **ncurses** package will be needed to link. On most systems, including Red Hat and SuSE, you should include the ncurses package. If Bacula's configure process finds the ncurses libraries, it will use those rather than the termcap library. On some systems, such as SuSE, the termcap library is not in the standard library directory. As a consequence, the option may be disabled or you may get an error message such as:

```
/usr/lib/gcc-lib/i586-suse-linux/3.3.1/.../ld:
cannot find -ltermcap
collect2: ld returned 1 exit status
```

while building the Bacula Console. In that case, you will need to set the **LDFLAGS** environment variable prior to building.

```
export LDFLAGS="-L/usr/lib/termcap"
```

The same library requirements apply if you wish to use the readline subroutines for command line editing and history or if you are using a MySQL library that requires encryption. If you need encryption, you can either export the appropriate additional library options as shown above or, alternatively, you can include them directly on the ./configure line as in:

```
LDFLAGS="-lssl -lcyrpto" \
   ./configure <your-options>
```

On some systems such as Mandriva, readline tends to gobble up prompts, which makes it totally useless. If this happens to you, use the disable option, or if you are using version 1.33 and above try using **--enable-conio** to use a built-in readline replacement. You will still need either the termcap or the ncurses library, but it is unlikely that the **conio** package will gobble up prompts.

readline is no longer supported after version 1.34. The code within Bacula remains, so it should be usable, and if users submit patches for it, we will be happy to apply them. However, due to the fact that each version of readline seems to be incompatible with previous versions, and that there are significant differences between systems, we can no longer afford to support it.

2.7 What Database to Use?

Before building Bacula you need to decide if you want to use SQLite, MySQL, or PostgreSQL. If you are not already running MySQL or PostgreSQL, you might want to start by testing with SQLite (not supported on Solaris). This will greatly simplify the setup for you because SQLite is compiled into Bacula an requires no administration. It performs well and is suitable for small to medium sized installations (maximum 10-20 machines). However, we should note that a number of users have had unexplained database corruption with SQLite. For that reason, we recommend that you install either MySQL or PostgreSQL for production work.

If you wish to use MySQL as the Bacula catalog, please see the Installing and Configuring MySQL chapter of this manual. You will need to install MySQL prior to continuing with the configuration of Bacula. MySQL is a high quality database that is very efficient and is suitable for any sized installation. It is slightly more complicated than SQLite to setup and administer because it has a number of sophisticated features such as userids and passwords. It runs as a separate process, is truly professional and can manage a database of any size.

If you wish to use PostgreSQL as the Bacula catalog, please see the Installing and Configuring PostgreSQL chapter of this manual. You will need to install PostgreSQL prior to continuing with the configuration of Bacula. PostgreSQL is very similar to MySQL, though it tends to be slightly more SQL92 compliant and has many more advanced features such as transactions, stored procedures, and the such. It requires a certain knowledge to install and maintain.

If you wish to use SQLite as the Bacula catalog, please see Installing and Configuring SQLite chapter of this manual. SQLite is not supported on Solaris.

2.8 Quick Start

There are a number of options and important considerations given below that you can skip for the moment if you have not had any problems building Bacula with a simplified configuration as shown above.

If the ./configure process is unable to find specific libraries (e.g. libintl, you should ensure that the appropriate package is installed on your system. Alternatively, if the package is installed in a non-standard location (as far as Bacula is concerned), then there is generally an option listed below (or listed with "./configure --help" that will permit you to specify the directory that should be searched. In other cases, there are options that will permit you to disable to feature (e.g. --disable-nls).

If you want to dive right into it, we recommend you skip to the next chapter, and run the example program. It will teach you a lot about Bacula and as an example can be installed into a single directory (for easy removal) and run as non-root. If you have any problems or when you want to do a real installation, come back to this chapter and read the details presented below.

2.9 Configure Options

The following command line options are available for **configure** to customize your installation.

-prefix=<patch> This option is meant to allow you to direct where the architecture independent files should be placed. However, we find this a somewhat vague concept, and so we have not implemented this option other than what ./configure does by default. As a consequence, we suggest that you avoid it. We have provided options that allow you to explicitly specify the directories for each of the major categories of installation files.

--sbindir=<binary-path> Defines where the Bacula binary (executable) files will be placed during a **make install** command.

--sysconfdir=<config-path> Defines where the Bacula configuration files should be placed during a **make install** command.

--mandir=<path> Note, as of Bacula version 1.39.14, the meaning of any path specified on this option is change from prior versions. It now specifies the top level man directory. Previously the mandir specified the full path to where you wanted the man files installed. The man files will be installed in gzip'ed format under mandir/man1 and mandir/man8 as appropriate. For the install to succeed you must have **gzip** installed on your system.

By default, Bacula will install the Unix man pages in /usr/share/man/man1 and /usr/share/man/man8. If you wish the man page to be installed in a different location, use this option to specify the path. Note, the main HTML and PDF Bacula documents are in a separate tar file that is not part of the source distribution.

--datadir=<path> If you translate Bacula or parts of Bacula into a different language you may specify the location of the po files using the **--datadir** option. You must manually install any po files as Bacula does not (yet) automatically do so.

--disable-ipv6

--enable-smartalloc This enables the inclusion of the Smartalloc orphaned buffer detection code. This option is highly recommended. Because we never build without this option, you may experience problems if it is not enabled. In this case, simply re-enable the option. We strongly recommend keeping this option enabled as it helps detect memory leaks. This configuration parameter is used while building Bacula

--enable-bat If you have Qt4 ¿= 4.3 installed on your computer including the libqt4 and libqt4-devel (libqt4-dev on Debian) libraries, and you want to use the Bacula Administration Tool (bat) GUI Console interface to Bacula, you must specify this option. Doing so will build everything in the **src/qt-console** directory. The build with enable-bat will work only with a full Bacula build (i.e. it will not work with a client-only build).

Qt4 is available on OpenSUSE 10.2, CentOS 5, Fedora, and Debian. If it is not available on your system, you can download the **depkgs-qt** package from the Bacula Source Forge download area and build it and the qwt package, both of which are needed to build bat. See the INSTALL file in that package for more details. In particular to use the Qt4 built by **depkgs-qt** you bf must source the file **qt4-paths**.

--with-qwt=<path> The qwt package is a graphics library for Qt. If it is included during the building of bat, you will get one extra graphical function. At the current time, we recommend not including this option when building bat. The path specified must be an absolute path and not relative.

The qwt package is available for download from the qwt project on Source Forge. If you wish, you may build and install it on your system (by default in /usr/lib). If you have done so, you would specify:

```
--with-qwt=/usr/lib/qwt-5.0.2
```

Alternatively, you can download the Bacula depkgs-qt package (currently version 28Jul09) and build it, then assuming that you have put it into a directory named bacula, you would specify:

```
--with-qwt=$HOME/bacula/depkgs-qt/qwt
```

Some packages such as Debian do not adhere to the standard of naming the library libqwt.a or libqwt.so, and you will either need to manually add a soft link to the name they use or use the depkgs version, which handles the naming correctly.

--enable-batch-insert This option enables batch inserts of the attribute records (default) in the catalog database, which is much faster (10 times or more) than without this option for large numbers of files. However, this option will automatically be disabled if your SQL libraries are not thread safe. If you find that batch mode is not enabled on your Bacula installation, then your database most likely does not support threads.

SQLite2 is not thread safe. Batch insert cannot be enabled when using SQLite2

On most systems, MySQL, PostgreSQL and SQLite3 are thread safe.

To verify that your PostgreSQL is thread safe, you can try this (change the path to point to your particular installed libpq.a; these commands were issued on FreeBSD 6.2):

```
$ nm /usr/local/lib/libpq.a | grep PQputCopyData
00001b08 T PQputCopyData
$ nm /usr/local/lib/libpq.a | grep mutex
         U pthread_mutex_lock
         U pthread_mutex_unlock
         U pthread_mutex_init
         U pthread_mutex_lock
         U pthread_mutex_unlock
```

The above example shows a libpq that contains the required function PQputCopyData and is thread enabled (i.e. the pthread_mutex* entries). If you do not see PQputCopyData, your version of PostgreSQL is too old to allow batch insert. If you do not see the mutex entries, then thread support has not been enabled. Our tests indicate you usually need to change the configuration options and recompile/reinstall the PostgreSQL client software to get thread support.

Bacula always links to the thread safe MySQL libraries.

As a default, Bacula runs SQLite3 with **PRAGMA synchronous=OFF** because it improves performance by more than 30 times. However, it increases the possibility of a corrupted database. If you want more security, please modify src/version.h appropriately (it should be obvious when you look at the file).

Running with Batch Insert turned on is recommended because it can significantly improve attribute insertion times. However, it does put a significantly larger part of the work on your SQL engine, so you may need to pay more attention to tuning it. In particular, Batch Insert can require large temporary table space, and consequently, the default location (often /tmp) may run out of space causing errors. For MySQL, the location is set in my.conf with "tmpdir". You may also want to increase the memory available to your SQL engine to further improve performance during Batch Inserts.

--enable-gnome If you have GNOME installed on your computer including the GNOME development libraries, and you want to use the GNOME GUI Console interface to Bacula, you must specify this option. Doing so will build everything in the **src/gnome2-console** directory.

--enable-bwx-console If you have wxWidgets installed on your computer and you want to use the wxWidgets GUI Console interface to Bacula, you must specify this option. Doing so will build everything in the **src/wx-console** directory. This could also be useful to users who want a GUI Console and don't want to install GNOME, as wxWidgets can work with GTK+, Motif or even X11 libraries.

--enable-tray-monitor If you have GTK installed on your computer, you run a graphical environment or a window manager compatible with the FreeDesktop system tray standard (like KDE and GNOME) and you want to use a GUI to monitor Bacula daemons, you must specify this option. Doing so will build everything in the **src/tray-monitor** directory. Note, due to restrictions on what can be linked with GPLed code, we were forced to remove the egg code that dealt with the tray icons and replace it by calls to the GTK+ API, and unfortunately, the tray icon API necessary was not implemented until GTK version 2.10 or later.

--enable-static-tools This option causes the linker to link the Storage daemon utility tools (**bls**, **bextract**, and **bscan**) statically. This permits using them without having the shared libraries loaded. If you have problems linking in the **src/stored** directory, make sure you have not enabled this option, or explicitly disable static linking by adding **--disable-static-tools**.

--enable-static-fd This option causes the make process to build a **static-bacula-fd** in addition to the standard File daemon. This static version will include statically linked libraries and is required for the Bare Metal recovery. This option is largely superseded by using **make static-bacula-fd** from with in the **src/filed** directory. Also, the **--enable-client-only** option described below is useful for just building a client so that all the other parts of the program are not compiled.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **--openssl** or **--with-python** on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

--enable-static-sd This option causes the make process to build a **static-bacula-sd** in addition to the standard Storage daemon. This static version will include statically linked libraries and could be useful during a Bare Metal recovery.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **--openssl** or **--with-python** on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

--enable-static-dir This option causes the make process to build a **static-bacula-dir** in addition to the standard Director. This static version will include statically linked libraries and could be useful during a Bare Metal recovery.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **--openssl** or **--with-python** on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

--enable-static-cons This option causes the make process to build a **static-console** and a **static-gnome-console** in addition to the standard console. This static version will include statically linked libraries and could be useful during a Bare Metal recovery.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **--openssl** or **--with-python** on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

--enable-client-only This option causes the make process to build only the File daemon and the libraries that it needs. None of the other daemons, storage tools, nor the console will be built. Likewise a **make install** will then only install the File daemon. To cause all daemons to be built, you will need to do a configuration without this option. This option greatly facilitates building a Client on a client only machine.

When linking a static binary, the linker needs the static versions of all the libraries that are used, so frequently users will experience linking errors when this option is used. The first thing to do is to make sure you have the static glibc library installed on your system. The second thing to do is the make sure you do not specify **--openssl** or **--with-python** on your ./configure statement as these options require additional libraries. You may be able to enable those options, but you will need to load additional static libraries.

--enable-build-dird This option causes the make process to build the Director and the Director's tools. By default, this option is on, but you may turn it off by using **--disable-build-dird** to prevent the Director from being built.

--enable-build-stored This option causes the make process to build the Storage daemon. By default, this option is on, but you may turn it off by using **--disable-build-stored** to prevent the Storage daemon from being built.

--enable-largefile This option (default) causes Bacula to be built with 64 bit file address support if it is available on your system. This permits Bacula to read and write files greater than 2 GBytes in size. You may disable this feature and revert to 32 bit file addresses by using **--disable-largefile**.

--disable-nls By default, Bacula uses the GNU Native Language Support (NLS) libraries. On some machines, these libraries may not be present or may not function correctly (especially on non-Linux implementations). In such cases, you may specify **--disable-nls** to disable use of those libraries. In such a case, Bacula will revert to using English.

--disable-ipv6 By default, Bacula enables IPv6 protocol. On some systems, the files for IPv6 may exist, but the functionality could be turned off in the kernel. In that case, in order to correctly build Bacula, you will explicitly need to use this option so that Bacula does not attempt to reference OS function calls that do not exist.

--with-sqlite=<sqlite-path> This enables use of the SQLite version 2.8.x database. The **sqlite-path** is not normally specified as Bacula looks for the necessary components in a standard location (**depkgs/sqlite**). See Installing and Configuring SQLite chapter of this manual for more details. SQLite is not supported on Solaris.

See the note below under the --with-postgresql item.

--with-sqlite3=<sqlite3-path> This enables use of the SQLite version 3.x database. The **sqlite3-path** is not normally specified as Bacula looks for the necessary components in a standard location (**depkgs/sqlite3**). See Installing and Configuring SQLite chapter of this manual for more details. SQLite3 is not supported on Solaris.

--with-mysql=<mysql-path> This enables building of the Catalog services for Bacula. It assumes that MySQL is running on your system, and expects it to be installed in the **mysql-path** that you specify. Normally, if MySQL is installed in a standard system location, you can simply use **--with-mysql** with no path specification. If you do use this option, please proceed to installing MySQL in the Installing and Configuring MySQL chapter before proceeding with the configuration.

See the note below under the --with-postgresql item.

--with-postgresql=<path> This provides an explicit path to the PostgreSQL libraries if Bacula cannot find it by default. Normally to build with PostgreSQL, you would simply use **--with-postgresql**.

Note, for Bacula to be configured properly, you must specify one of the four database options supported. That is: --with-sqlite, --with-sqlite3, --with-mysql, or --with-postgresql, otherwise the ./configure will fail.

--with-openssl=<path> This configuration option is necessary if you want to enable TLS (ssl), which encrypts the communications within Bacula or if you want to use File Daemon PKI data encryption. Normally, the **path** specification is not necessary since the configuration searches for the OpenSSL libraries in standard system locations. Enabling OpenSSL in Bacula permits secure communications between the daemons and/or data encryption in the File daemon. For more information on using TLS, please see the Bacula TLS – Communications Encryption chapter of this manual. For more information on using PKI data encryption, please see the Bacula PKI – Data Encryption chapter of this manual.

--with-python=<path> This option enables Bacula support for Python. If no path is supplied, configure will search the standard library locations for Python 2.2, 2.3, 2.4, or 2.5. If it cannot find the library, you will need to supply a path to your Python library directory. Please see the Python chapter for the details of using Python scripting.

--with-libintl-prefix=<DIR> This option may be used to tell Bacula to search DIR/include and DIR/lib for the libintl headers and libraries needed for Native Language Support (NLS).

--enable-conio Tells Bacula to enable building the small, light weight readline replacement routine. It is generally much easier to configure than readline, although, like readline, it needs either the termcap or ncurses library.

--with-readline=<readline-path> Tells Bacula where **readline** is installed. Normally, Bacula will find readline if it is in a standard library. If it is not found and no --with-readline is specified, readline will be disabled. This option affects the Bacula build. Readline provides the Console program with a command line history and editing capability and is no longer supported, so you are on your own if you have problems.

--enable-readline Tells Bacula to enable readline support. It is normally disabled due to the large number of configuration problems and the fact that the package seems to change in incompatible ways from version to version.

--with-tcp-wrappers=<path> This specifies that you want TCP wrappers (man hosts_access(5)) compiled in. The path is optional since Bacula will normally find the libraries in the standard locations. This option affects the Bacula build. In specifying your restrictions in the **/etc/hosts.allow** or **/etc/hosts.deny** files, do not use the **twist** option (hosts_options(5)) or the Bacula process will be terminated. Note, when setting up your **/etc/hosts.allow** or **/etc/hosts.deny**, you must identify the Bacula daemon in question with the name you give it in your conf file rather than the name of the executable.

For more information on configuring and testing TCP wrappers, please see the Configuring and Testing TCP Wrappers section in the Security Chapter.

On SuSE, the libwrappers libraries needed to link Bacula are contained in the tcpd-devel package. On Red Hat, the package is named tcp_wrappers.

--with-archivedir=<path> The directory used for disk-based backups. Default value is /tmp. This parameter sets the default values in the bacula-dir.conf and bacula-sd.conf configuration files. For example, it sets the Where directive for the default restore job and the Archive Device directive for the FileStorage device.

This option is designed primarily for use in regression testing. Most users can safely ignore this option.

--with-working-dir=<working-directory-path> This option is mandatory and specifies a directory into which Bacula may safely place files that will remain between Bacula executions. For example, if the internal database is used, Bacula will keep those files in this directory. This option is only used to modify the daemon configuration files. You may also accomplish the same thing by directly editing them later. The working directory is not automatically created by the install process, so you must ensure that it exists before using Bacula for the first time.

--with-base-port=<port=number> In order to run, Bacula needs three TCP/IP ports (one for the Bacula Console, one for the Storage daemon, and one for the File daemon). The **--with-baseport** option will automatically assign three ports beginning at the base port address specified. You may also change the port number in the resulting configuration files. However, you need to take care that the numbers correspond correctly in each of the three daemon configuration files. The default base port is 9101, which assigns ports 9101 through 9103. These ports (9101, 9102, and 9103) have been officially assigned to Bacula by IANA. This option is only used to modify the daemon configuration files. You may also accomplish the same thing by directly editing them later.

--with-dump-email=<email-address> This option specifies the email address where any core dumps should be set. This option is normally only used by developers.

--with-pid-dir=<PATH> This specifies where Bacula should place the process id file during execution. The default is: **/var/run**. This directory is not created by the install process, so you must ensure that it exists before using Bacula the first time.

--with-subsys-dir=<PATH> This specifies where Bacula should place the subsystem lock file during execution. The default is **/var/run/subsys**. Please make sure that you do not specify the same directory for this directory and for the **sbindir** directory. This directory is used only within the autostart scripts. The subsys directory is not created by the Bacula install, so you must be sure to create it before using Bacula.

--with-dir-password=<Password> This option allows you to specify the password used to access the Director (normally from the Console program). If it is not specified, configure will automatically create a random password.

--with-fd-password=<Password> This option allows you to specify the password used to access the File daemon (normally called from the Director). If it is not specified, configure will automatically create a random password.

--with-sd-password=<Password> This option allows you to specify the password used to access the Storage daemon (normally called from the Director). If it is not specified, configure will automatically create a random password.

--with-dir-user=<User> This option allows you to specify the Userid used to run the Director. The Director must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the UserId specified on this option. If you specify this option, you must create the User prior to running **make install**, because the working directory owner will be set to **User**.

--with-dir-group=<Group> This option allows you to specify the GroupId used to run the Director. The Director must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the GroupId specified on this option. If you specify this option, you must create the Group prior to running **make install**, because the working directory group will be set to **Group**.

--with-sd-user=<User> This option allows you to specify the Userid used to run the Storage daemon. The Storage daemon must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the UserId specified on this option. If you use this option, you will need to take care that the Storage daemon has access to all the devices (tape drives, ...) that it needs.

--with-sd-group=<Group> This option allows you to specify the GroupId used to run the Storage daemon. The Storage daemon must be started as root, but doesn't need to run as root, and after doing preliminary initializations, it can "drop" to the GroupId specified on this option.

--with-fd-user=<User> This option allows you to specify the Userid used to run the File daemon. The File daemon must be started as root, and in most cases, it needs to run as root, so this option is used only in very special cases, after doing preliminary initializations, it can "drop" to the UserId specified on this option.

--with-fd-group=<Group> This option allows you to specify the GroupId used to run the File daemon. The File daemon must be started as root, and in most cases, it must be run as root, however, after doing preliminary initializations, it can "drop" to the GroupId specified on this option.

--with-mon-dir-password=<Password> This option allows you to specify the password used to access the Directory from the monitor. If it is not specified, configure will automatically create a random password.

--with-mon-fd-password=<Password> This option allows you to specify the password used to access the File daemon from the Monitor. If it is not specified, configure will automatically create a random password.

--with-mon-sd-password=<Password> This option allows you to specify the password used to access the Storage daemon from the Monitor. If it is not specified, configure will automatically create a random password.

--with-db-name=<database-name> This option allows you to specify the database name to be used in the conf files. The default is bacula.

--with-db-user=<database-user> This option allows you to specify the database user name to be used in the conf files. The default is bacula.

Note, many other options are presented when you do a **./configure --help**, but they are not implemented.

2.10 Recommended Options for Most Systems

For most systems, we recommend starting with the following options:

```
./configure \
  --enable-smartalloc \
  --sbindir=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working \
  --with-mysql=$HOME/mysql \
  --with-working-dir=$HOME/bacula/working
```

If you want to install Bacula in an installation directory rather than run it out of the build directory (as developers will do most of the time), you should also include the --sbindir and --sysconfdir options with appropriate paths. Neither are necessary if you do not use "make install" as is the case for most development work. The install process will create the sbindir and sysconfdir if they do not exist, but it will not automatically create the pid-dir, subsys-dir, or working-dir, so you must ensure that they exist before running Bacula for the first time.

2.11 Red Hat

Using SQLite:

```
CFLAGS="-g -Wall" ./configure \
  --sbindir=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --enable-smartalloc \
  --with-sqlite=$HOME/bacula/depkgs/sqlite \
  --with-working-dir=$HOME/bacula/working \
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working \
  --enable-bat \
  --with-qwt=$HOME/bacula/depkgs/qwt \
  --enable-conio
```

or

```
CFLAGS="-g -Wall" ./configure \
  --sbindir=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --enable-smartalloc \
  --with-mysql=$HOME/mysql \
  --with-working-dir=$HOME/bacula/working
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working
  --enable-gnome \
  --enable-conio
```

or finally, a completely traditional Red Hat Linux install:

```
CFLAGS="-g -Wall" ./configure \
  --sbindir=/usr/sbin \
  --sysconfdir=/etc/bacula \
  --with-scriptdir=/etc/bacula \
  --enable-smartalloc \
  --enable-bat \
  --with-qwt=$HOME/bacula/depkgs/qwt \
  --with-mysql \
  --with-working-dir=/var/bacula \
  --with-pid-dir=/var/run \
  --enable-conio
```

Note, Bacula assumes that /var/bacula, /var/run, and /var/lock/subsys exist so it will not automatically create them during the install process.

2.12 Solaris

To build Bacula from source, you will need the following installed on your system (they are not by default): libiconv, gcc 3.3.2, stdc++, libgcc (for stdc++ and gcc_s libraries), make 3.8 or later.

You will probably also need to: Add /usr/local/bin to PATH and Add /usr/ccs/bin to PATH for ar.

It is possible to build Bacula on Solaris with the Solaris compiler, but we recommend using GNU C++ if possible.

A typical configuration command might look like:

```
#!/bin/sh
```

```
CFLAGS="-g" ./configure \
  --sbindir=$HOME/bacula/bin \
  --sysconfdir=$HOME/bacula/bin \
  --with-mysql=$HOME/mysql \
  --enable-smartalloc \
  --with-pid-dir=$HOME/bacula/bin/working \
  --with-subsys-dir=$HOME/bacula/bin/working \
  --with-working-dir=$HOME/bacula/working
```

As mentioned above, the install process will create the sbindir and sysconfdir if they do not exist, but it will not automatically create the pid-dir, subsys-dir, or working-dir, so you must ensure that they exist before running Bacula for the first time.

Note, you may need to install the following packages to build Bacula from source:

```
SUNWbinutils,
SUNWarc,
SUNWhea,
SUNWGcc,
SUNWGnutls
SUNWGnutls-devel
SUNWGmake
SUNWgccruntime
SUNWlibgcrypt
SUNWzlib
SUNWzlibs
SUNWbinutilsS
SUNWGmakeS
SUNWlibm

export
PATH=/usr/bin::/usr/ccs/bin:/etc:/usr/openwin/bin:/usr/local/bin:/usr/sfw/bin:/opt/sfw/bin:/usr/ucb:/usr/sbin
```

If you have installed special software not normally in the Solaris libraries, such as OpenSSL, or the packages shown above, then you may need to add **/usr/sfw/lib** to the library search path. Probably the simplest way to do so is to run:

```
setenv LDFLAGS "-L/usr/sfw/lib -R/usr/sfw/lib"
```

Prior to running the ./configure command.

Alternatively, you can set the LD_LIBARY_PATH and/or the LD_RUN_PATH environment variables appropriately.

It is also possible to use the **crle** program to set the library search path. However, this should be used with caution.

2.13 FreeBSD

Please see: The FreeBSD Diary for a detailed description on how to make Bacula work on your system. In addition, users of FreeBSD prior to 4.9-STABLE dated Mon Dec 29 15:18:01 2003 UTC who plan to use tape devices, please see the Tape Testing Chapter of this manual for **important** information on how to configure your tape drive for compatibility with Bacula.

If you are using Bacula with MySQL, you should take care to compile MySQL with FreeBSD native threads rather than LinuxThreads, since Bacula is normally built with FreeBSD native threads rather than LinuxTreads. Mixing the two will probably not work.

2.14 Win32

To install the binary Win32 version of the File daemon please see the Win32 Installation Chapter in this document.

2.15 One File Configure Script

The following script could be used if you want to put everything in a single file:

```
#!/bin/sh
CFLAGS="-g -Wall" \
  ./configure \
    --sbindir=$HOME/bacula/bin \
    --sysconfdir=$HOME/bacula/bin \
    --mandir=$HOME/bacula/bin \
    --enable-smartalloc \
    --enable-gnome \
    --enable-bat \
    --with-qwt=$HOME/bacula/depkgs/qwt \
    --enable-bwx-console \
    --enable-tray-monitor \
    --with-pid-dir=$HOME/bacula/bin/working \
    --with-subsys-dir=$HOME/bacula/bin/working \
    --with-mysql \
    --with-working-dir=$HOME/bacula/bin/working \
    --with-dump-email=$USER@your-site.com \
    --with-job-email=$USER@your-site.com \
    --with-smtp-host=mail.your-site.com
exit 0
```

You may also want to put the following entries in your **/etc/services** file as it will make viewing the connections made by Bacula easier to recognize (i.e. netstat -a):

```
bacula-dir      9101/tcp
bacula-fd       9102/tcp
bacula-sd       9103/tcp
```

2.16 Installing Bacula

Before setting up your configuration files, you will want to install Bacula in its final location. Simply enter:

```
make install
```

If you have previously installed Bacula, the old binaries will be overwritten, but the old configuration files will remain unchanged, and the "new" configuration files will be appended with a **.new**. Generally if you have previously installed and run Bacula you will want to discard or ignore the configuration files with the appended **.new**.

2.17 Building a File Daemon or Client

If you run the Director and the Storage daemon on one machine and you wish to back up another machine, you must have a copy of the File daemon for that machine. If the machine and the Operating System are identical, you can simply copy the Bacula File daemon binary file **bacula-fd** as well as its configuration file **bacula-fd.conf** then modify the name and password in the conf file to be unique. Be sure to make corresponding additions to the Director's configuration file (**bacula-dir.conf**).

If the architecture or the OS level are different, you will need to build a File daemon on the Client machine. To do so, you can use the same **./configure** command as you did for your main program, starting either from a fresh copy of the source tree, or using **make distclean** before the **./configure**.

Since the File daemon does not access the Catalog database, you can remove the **--with-mysql** or **--with-sqlite** options, then add **--enable-client-only**. This will compile only the necessary libraries and the client programs and thus avoids the necessity of installing one or another of those database programs to build the File daemon. With the above option, you simply enter **make** and just the client will be built.

2.18 Auto Starting the Daemons

If you wish the daemons to be automatically started and stopped when your system is booted (a good idea), one more step is necessary. First, the ./configure process must recognize your system – that is it must be a supported platform and not **unknown**, then you must install the platform dependent files by doing:

```
(become root)
make install-autostart
```

Please note, that the auto-start feature is implemented only on systems that we officially support (currently, FreeBSD, Red Hat/Fedora Linux, and Solaris), and has only been fully tested on Fedora Linux.

The **make install-autostart** will cause the appropriate startup scripts to be installed with the necessary symbolic links. On Red Hat/Fedora Linux systems, these scripts reside in **/etc/rc.d/init.d/bacula-dir /etc/rc.d/init.d/bacula-fd**, and **/etc/rc.d/init.d/bacula-sd**. However the exact location depends on what operating system you are using.

If you only wish to install the File daemon, you may do so with:

```
make install-autostart-fd
```

2.19 Other Make Notes

To simply build a new executable in any directory, enter:

```
make
```

To clean out all the objects and binaries (including the files named 1, 2, or 3, which are development temporary files), enter:

```
make clean
```

To really clean out everything for distribution, enter:

```
make distclean
```

note, this cleans out the Makefiles and is normally done from the top level directory to prepare for distribution of the source. To recover from this state, you must redo the **./configure** in the top level directory, since all the Makefiles will be deleted.

To add a new file in a subdirectory, edit the Makefile.in in that directory, then simply do a **make**. In most cases, the make will rebuild the Makefile from the new Makefile.in. In some case, you may need to issue the **make** a second time. In extreme cases, cd to the top level directory and enter: **make Makefiles**.

To add dependencies:

```
make depend
```

The **make depend** appends the header file dependencies for each of the object files to Makefile and Makefile.in. This command should be done in each directory where you change the dependencies. Normally, it only needs to be run when you add or delete source or header files. **make depend** is normally automatically invoked during the configuration process.

To install:

```
make install
```

This not normally done if you are developing Bacula, but is used if you are going to run it to backup your system.

After doing a **make install** the following files will be installed on your system (more or less). The exact files and location (directory) for each file depends on your **./configure** command (e.g. bgnome-console and bgnome-console.conf are not installed if you do not configure GNOME. Also, if you are using SQLite instead of MySQL, some of the files will be different).

NOTE: it is quite probable that this list is out of date. But it is a starting point.

```
bacula
bacula-dir
bacula-dir.conf
bacula-fd
bacula-fd.conf
bacula-sd
bacula-sd.conf
bacula-tray-monitor
tray-monitor.conf
bextract
bls
bscan
btape
btraceback
btraceback.gdb
bconsole
bconsole.conf
create_mysql_database
dbcheck
delete_catalog_backup
drop_bacula_tables
drop_mysql_tables
bgnome-console
bgnome-console.conf
make_bacula_tables
make_catalog_backup
make_mysql_tables
mtx-changer
query.sql
bsmtp
startmysql
stopmysql
bwx-console
bwx-console.conf
9 man pages
```

2.20 Installing Tray Monitor

The Tray Monitor is already installed if you used the **--enable-tray-monitor** configure option and ran **make install**.

As you don't run your graphical environment as root (if you do, you should change that bad habit), don't forget to allow your user to read **tray-monitor.conf**, and to execute **bacula-tray-monitor** (this is not a security issue).

Then log into your graphical environment (KDE, GNOME or something else), run **bacula-tray-monitor** as your user, and see if a cassette icon appears somewhere on the screen, usually on the task bar. If it doesn't, follow the instructions below related to your environment or window manager.

2.20.1 GNOME

System tray, or notification area if you use the GNOME terminology, has been supported in GNOME since version 2.2. To activate it, right-click on one of your panels, open the menu **Add to this Panel**, then **Utility** and finally click on **Notification Area**.

2.20.2 KDE

System tray has been supported in KDE since version 3.1. To activate it, right-click on one of your panels, open the menu **Add**, then **Applet** and finally click on **System Tray**.

2.20.3 Other window managers

Read the documentation to know if the Freedesktop system tray standard is supported by your window manager, and if applicable, how to activate it.

2.21 Modifying the Bacula Configuration Files

See the chapter Configuring Bacula in this manual for instructions on how to set Bacula configuration files.

Chapter 3

Critical Items to Implement Before Production

We recommend you take your time before implementing a production a Bacula backup system since Bacula is a rather complex program, and if you make a mistake, you may suddenly find that you cannot restore your files in case of a disaster. This is especially true if you have not previously used a major backup product.

If you follow the instructions in this chapter, you will have covered most of the major problems that can occur. It goes without saying that if you ever find that we have left out an important point, please inform us, so that we can document it to the benefit of everyone.

3.1 Critical Items

The following assumes that you have installed Bacula, you more or less understand it, you have at least worked through the tutorial or have equivalent experience, and that you have set up a basic production configuration. If you haven't done the above, please do so and then come back here. The following is a sort of checklist that points with perhaps a brief explanation of why you should do it. In most cases, you will find the details elsewhere in the manual. The order is more or less the order you would use in setting up a production system (if you already are in production, use the checklist anyway).

- Test your tape drive for compatibility with Bacula by using the test command in the btape program.
- Better than doing the above is to walk through the nine steps in the Tape Testing chapter of the manual. It may take you a bit of time, but it will eliminate surprises.
- Test the end of tape handling of your tape drive by using the fill command in the btape program.
- If you are using a Linux 2.4 kernel, make sure that /lib/tls is disabled. Bacula does not work with this library. See the second point under Supported Operating Systems.
- Do at least one restore of files. If you backup multiple OS types (Linux, Solaris, HP, MacOS, FreeBSD, Win32, ...), restore files from each system type. The Restoring Files chapter shows you how.
- Write a bootstrap file to a separate system for each backup job. The Write Bootstrap directive is described in the Director Configuration chapter of the manual, and more details are available in the Bootstrap File chapter. Also, the default bacula-dir.conf comes with a Write Bootstrap directive defined. This allows you to recover the state of your system as of the last backup.
- Backup your catalog. An example of this is found in the default bacula-dir.conf file. The backup script is installed by default and should handle any database, though you may want to make your own local modifications. See also Backing Up Your Bacula Database - Security Considerations for more information.

- Write a bootstrap file for the catalog. An example of this is found in the default bacula-dir.conf file. This will allow you to quickly restore your catalog in the event it is wiped out – otherwise it is many excruciating hours of work.
- Make a copy of the bacula-dir.conf, bacula-sd.conf, and bacula-fd.conf files that you are using on your server. Put it in a safe place (on another machine) as these files can be difficult to reconstruct if your server dies.
- Make a Bacula Rescue CDROM! See the Disaster Recovery Using a Bacula Rescue CDROM chapter. It is trivial to make such a CDROM, and it can make system recovery in the event of a lost hard disk infinitely easier.
- Bacula assumes all filenames are in UTF-8 format. This is important when saving the filenames to the catalog. For Win32 machine, Bacula will automatically convert from Unicode to UTF-8, but on Unix, Linux, *BSD, and MacOS X machines, you must explicitly ensure that your locale is set properly. Typically this means that the bf LANG environment variable must end in **.UTF-8**. An full example is **en_US.UTF-8**. The exact syntax may vary a bit from OS to OS, and exactly how you define it will also vary.

  On most modern Win32 machines, you can edit the conf files with **notepad** and choose output encoding UTF-8.

3.2 Recommended Items

Although these items may not be critical, they are recommended and will help you avoid problems.

- Read the Quick Start Guide to Bacula
- After installing and experimenting with Bacula, read and work carefully through the examples in the Tutorial chapter of this manual.
- Learn what each of the Bacula Utility Programs does.
- Set up reasonable retention periods so that your catalog does not grow to be too big. See the following three chapters:
Recycling your Volumes,
Basic Volume Management,
Using Pools to Manage Volumes.
- Perform a bare metal recovery using the Bacula Rescue CDROM. See the Disaster Recovery Using a Bacula Rescue CDROM chapter.

If you absolutely must implement a system where you write a different tape each night and take it offsite in the morning. We recommend that you do several things:

- Write a bootstrap file of your backed up data and a bootstrap file of your catalog backup to a floppy disk or a CDROM, and take that with the tape. If this is not possible, try to write those files to another computer or offsite computer, or send them as email to a friend. If none of that is possible, at least print the bootstrap files and take that offsite with the tape. Having the bootstrap files will make recovery much easier.
- It is better not to force Bacula to load a particular tape each day. Instead, let Bacula choose the tape. If you need to know what tape to mount, you can print a list of recycled and appendable tapes daily, and select any tape from that list. Bacula may propose a particular tape for use that it considers optimal, but it will accept any valid tape from the correct pool.

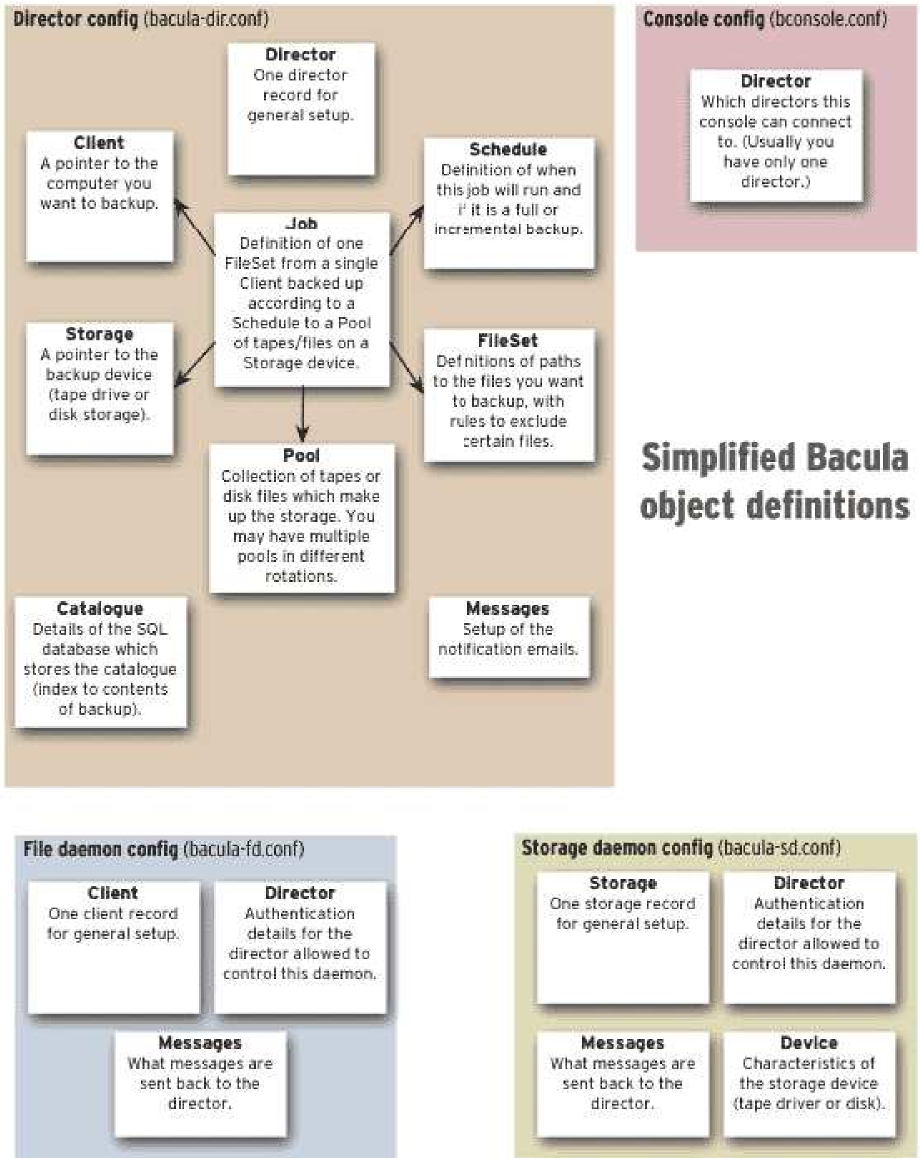
Chapter 4

Customizing the Configuration Files

When each of the Bacula programs starts, it reads a configuration file specified on the command line or the default **bacula-dir.conf**, **bacula-fd.conf**, **bacula-sd.conf**, or **console.conf** for the Director daemon, the File daemon, the Storage daemon, and the Console program respectively.

Each service (Director, Client, Storage, Console) has its own configuration file containing a set of Resource definitions. These resources are very similar from one service to another, but may contain different directives (records) depending on the service. For example, in the Director's resource file, the **Director** resource defines the name of the Director, a number of global Director parameters and his password. In the File daemon configuration file, the **Director** resource specifies which Directors are permitted to use the File daemon.

Before running Bacula for the first time, you must customize the configuration files for each daemon. Default configuration files will have been created by the installation process, but you will need to modify them to correspond to your system. An overall view of the resources can be seen in the following:



4.1 Character Sets

Bacula is designed to handle most character sets of the world, US ASCII, German, French, Chinese, ... However, it does this by encoding everything in UTF-8, and it expects all configuration files (including those read on Win32 machines) to be in UTF-8 format. UTF-8 is typically the default on Linux machines, but not on all Unix machines, nor on Windows, so you must take some care to ensure that your locale is set properly before starting Bacula.

To ensure that Bacula configuration files can be correctly read including foreign characters the bf LANG environment variable must end in **.UTF-8**. An full example is **en_US.UTF-8**. The exact syntax may vary a bit from OS to OS, and exactly how you define it will also vary. On most newer Win32 machines, you can use **notepad** to edit the conf files, then choose output encoding UTF-8.

Bacula assumes that all filenames are in UTF-8 format on Linux and Unix machines. On Win32 they are in Unicode (UTF-16), and will be automatically converted to UTF-8 format.

4.2 Resource Directive Format

Although, you won't need to know the details of all the directives a basic knowledge of Bacula resource directives is essential. Each directive contained within the resource (within the braces) is composed of a keyword followed by an equal sign (=) followed by one or more values. The keywords must be one of the known Bacula resource record keywords, and it may be composed of upper or lower case characters and spaces.

Each resource definition MUST contain a Name directive, and may optionally contain a Description directive. The Name directive is used to uniquely identify the resource. The Description directive is (will be) used during display of the Resource to provide easier human recognition. For example:

```
Director {
  Name = "MyDir"
  Description = "Main Bacula Director"
  WorkingDirectory = "$HOME/bacula/bin/working"
}
```

Defines the Director resource with the name "MyDir" and a working directory $HOME/bacula/bin/working. In general, if you want spaces in a name to the right of the first equal sign (=), you must enclose that name within double quotes. Otherwise quotes are not generally necessary because once defined, quoted strings and unquoted strings are all equal.

4.2.1 Comments

When reading the configuration file, blank lines are ignored and everything after a hash sign (#) until the end of the line is taken to be a comment. A semicolon (;) is a logical end of line, and anything after the semicolon is considered as the next statement. If a statement appears on a line by itself, a semicolon is not necessary to terminate it, so generally in the examples in this manual, you will not see many semicolons.

4.2.2 Upper and Lower Case and Spaces

Case (upper/lower) and spaces are totally ignored in the resource directive keywords (the part before the equal sign).

Within the keyword (i.e. before the equal sign), spaces are not significant. Thus the keywords: **name**, **Name**, and **N a m e** are all identical.

Spaces after the equal sign and before the first character of the value are ignored.

In general, spaces within a value are significant (not ignored), and if the value is a name, you must enclose the name in double quotes for the spaces to be accepted. Names may contain up to 127 characters. Currently, a name may contain any ASCII character. Within a quoted string, any character following a backslash (\) is taken as itself (handy for inserting backslashes and double quotes (")).

Please note, however, that Bacula resource names as well as certain other names (e.g. Volume names) must contain only letters (including ISO accented letters), numbers, and a few special characters (space, underscore, ...). All other characters and punctuation are invalid.

4.2.3 Including other Configuration Files

If you wish to break your configuration file into smaller pieces, you can do so by including other files using the syntax @**filename** where **filename** is the full path and filename of another file. The @filename specification can be given anywhere a primitive token would appear.

4.2.4 Recognized Primitive Data Types

When parsing the resource directives, Bacula classifies the data according to the types listed below. The first time you read this, it may appear a bit overwhelming, but in reality, it is all pretty logical and straightforward.

name A keyword or name consisting of alphanumeric characters, including the hyphen, underscore, and dollar characters. The first character of a **name** must be a letter. A name has a maximum length currently set to 127 bytes. Typically keywords appear on the left side of an equal (i.e. they are Bacula keywords – i.e. Resource names or directive names). Keywords may not be quoted.

name-string A name-string is similar to a name, except that the name may be quoted and can thus contain additional characters including spaces. Name strings are limited to 127 characters in length. Name strings are typically used on the right side of an equal (i.e. they are values to be associated with a keyword).

string A quoted string containing virtually any character including spaces, or a non-quoted string. A string may be of any length. Strings are typically values that correspond to filenames, directories, or system command names. A backslash (\) turns the next character into itself, so to include a double quote in a string, you precede the double quote with a backslash. Likewise to include a backslash.

directory A directory is either a quoted or non-quoted string. A directory will be passed to your standard shell for expansion when it is scanned. Thus constructs such as **$HOME** are interpreted to be their correct values.

password This is a Bacula password and it is stored internally in MD5 hashed format.

integer A 32 bit integer value. It may be positive or negative.

positive integer A 32 bit positive integer value.

long integer A 64 bit integer value. Typically these are values such as bytes that can exceed 4 billion and thus require a 64 bit value.

yes|no Either a **yes** or a **no**.

size A size specified as bytes. Typically, this is a floating point scientific input format followed by an optional modifier. The floating point input is stored as a 64 bit integer value. If a modifier is present, it must immediately follow the value with no intervening spaces. The following modifiers are permitted:

- **k** 1,024 (kilobytes)
- **kb** 1,000 (kilobytes)
- **m** 1,048,576 (megabytes)
- **mb** 1,000,000 (megabytes)
- **g** 1,073,741,824 (gigabytes)
- **gb** 1,000,000,000 (gigabytes)

time A time or duration specified in seconds. The time is stored internally as a 64 bit integer value, but it is specified in two parts: a number part and a modifier part. The number can be an integer or a floating point number. If it is entered in floating point notation, it will be rounded to the nearest integer. The modifier is mandatory and follows the number part, either with or without intervening spaces. The following modifiers are permitted:

- **seconds** seconds
- **minutes** minutes (60 seconds)
- **hours** hours (3600 seconds)
- **days** days (3600*24 seconds)
- **weeks** weeks (3600*24*7 seconds)
- **months** months (3600*24*30 seconds)
- **quarters** quarters (3600*24*91 seconds)

years years (3600*24*365 seconds)

Any abbreviation of these modifiers is also permitted (i.e. **seconds** may be specified as **sec** or **s**). A specification of **m** will be taken as months.

The specification of a time may have as many number/modifier parts as you wish. For example:

```
1 week 2 days 3 hours 10 mins
1 month 2 days 30 sec
```

are valid date specifications.

4.3 Resource Types

The following table lists all current Bacula resource types. It shows what resources must be defined for each service (daemon). The default configuration files will already contain at least one example of each permitted resource, so you need not worry about creating all these kinds of resources from scratch.

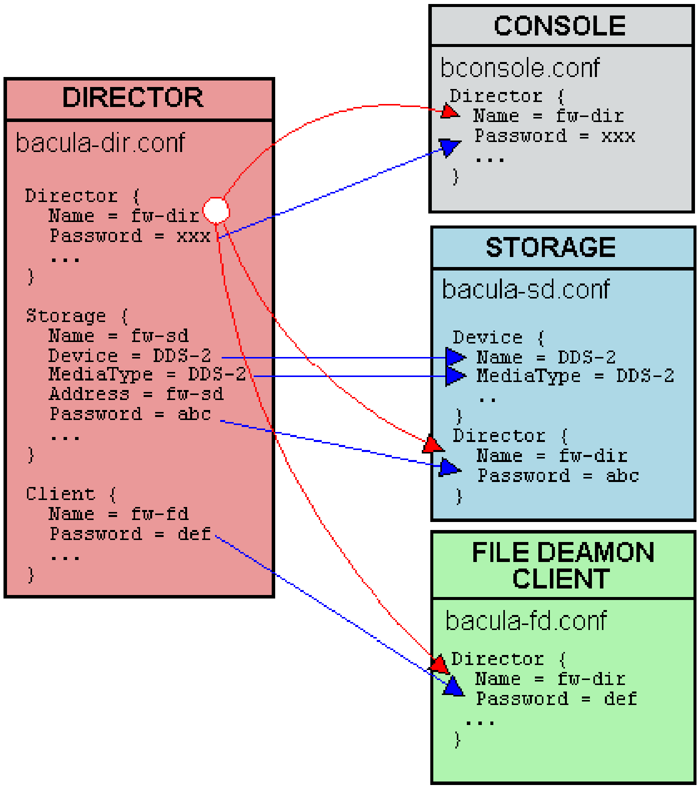
Resource	Director	Client	Storage	Console
Autochanger	No	No	Yes	No
Catalog	Yes	No	No	No
Client	Yes	Yes	No	No
Console	Yes	No	No	Yes
Device	No	No	Yes	No
Director	Yes	Yes	Yes	Yes
FileSet	Yes	No	No	No
Job	Yes	No	No	No
JobDefs	Yes	No	No	No
Message	Yes	Yes	Yes	No
Pool	Yes	No	No	No
Schedule	Yes	No	No	No
Storage	Yes	No	Yes	No

4.4 Names, Passwords and Authorization

In order for one daemon to contact another daemon, it must authorize itself with a password. In most cases, the password corresponds to a particular name, so both the name and the password must match to be authorized. Passwords are plain text, any text. They are not generated by any special process; just use random text.

The default configuration files are automatically defined for correct authorization with random passwords. If you add to or modify these files, you will need to take care to keep them consistent.

Here is sort of a picture of what names/passwords in which files/Resources must match up:



In the left column, you will find the Director, Storage, and Client resources, with their names and passwords – these are all in **bacula-dir.conf**. In the right column are where the corresponding values should be found in the Console, Storage daemon (SD), and File daemon (FD) configuration files.

Please note that the Address, **fd-sd**, that appears in the Storage resource of the Director, preceded with and asterisk in the above example, is passed to the File daemon in symbolic form. The File daemon then resolves it to an IP address. For this reason, you must use either an IP address or a fully qualified name. A name such as **localhost**, not being a fully qualified name, will resolve in the File daemon to the localhost of the File daemon, which is most likely not what is desired. The password used for the File daemon to authorize with the Storage daemon is a temporary password unique to each Job created by the daemons and is not specified in any .conf file.

4.5 Detailed Information for each Daemon

The details of each Resource and the directives permitted therein are described in the following chapters.

The following configuration files must be defined:

- Console – to define the resources for the Console program (user interface to the Director). It defines which Directors are available so that you may interact with them.
- Director – to define the resources necessary for the Director. You define all the Clients and Storage daemons that you use in this configuration file.
- Client – to define the resources for each client to be backed up. That is, you will have a separate Client resource file on each machine that runs a File daemon.
- Storage – to define the resources to be used by each Storage daemon. Normally, you will have a single Storage daemon that controls your tape drive or tape drives. However, if you have tape drives on several machines, you will have at least one Storage daemon per machine.

Chapter 5

Configuring the Director

Of all the configuration files needed to run **Bacula**, the Director's is the most complicated, and the one that you will need to modify the most often as you add clients or modify the FileSets.

For a general discussion of configuration files and resources including the data types recognized by **Bacula**. Please see the Configuration chapter of this manual.

5.1 Director Resource Types

Director resource type may be one of the following:

Job, JobDefs, Client, Storage, Catalog, Schedule, FileSet, Pool, Director, or Messages. We present them here in the most logical order for defining them:

Note, everything revolves around a job and is tied to a job in one way or another.

- Director – to define the Director's name and its access password used for authenticating the Console program. Only a single Director resource definition may appear in the Director's configuration file. If you have either **/dev/random** or **bc** on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.
- Job – to define the backup/restore Jobs and to tie together the Client, FileSet and Schedule resources to be used for each Job. Normally, you will Jobs of different names corresponding to each client (i.e. one Job per client, but a different one with a different name for each client).
- JobDefs – optional resource for providing defaults for Job resources.
- Schedule – to define when a Job is to be automatically run by **Bacula's** internal scheduler. You may have any number of Schedules, but each job will reference only one.
- FileSet – to define the set of files to be backed up for each Client. You may have any number of FileSets but each Job will reference only one.
- Client – to define what Client is to be backed up. You will generally have multiple Client definitions. Each Job will reference only a single client.
- Storage – to define on what physical device the Volumes should be mounted. You may have one or more Storage definitions.
- Pool – to define the pool of Volumes that can be used for a particular Job. Most people use a single default Pool. However, if you have a large number of clients or volumes, you may want to have multiple Pools. Pools allow you to restrict a Job (or a Client) to use only a particular set of Volumes.

- Catalog – to define in what database to keep the list of files and the Volume names where they are backed up. Most people only use a single catalog. However, if you want to scale the Director to many clients, multiple catalogs can be helpful. Multiple catalogs require a bit more management because in general you must know what catalog contains what data. Currently, all Pools are defined in each catalog. This restriction will be removed in a later release.
- Messages – to define where error and information messages are to be sent or logged. You may define multiple different message resources and hence direct particular classes of messages to different users or locations (files, ...).

5.2 The Director Resource

The Director resource defines the attributes of the Directors running on the network. In the current implementation, there is only a single Director resource, but the final design will contain multiple Directors to maintain index and media database redundancy.

Director Start of the Director resource. One and only one director resource must be supplied.

Name = <name> The director name used by the system administrator. This directive is required.

Description = <text> The text field contains a description of the Director that will be displayed in the graphical user interface. This directive is optional.

Password = <UA-password> Specifies the password that must be supplied for the default Bacula Console to be authorized. The same password must appear in the **Director** resource of the Console configuration file. For added security, the password is never passed across the network but instead a challenge response hash code created with the password. This directive is required. If you have either **/dev/random** or **bc** on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank and you must manually supply it.

The password is plain text. It is not generated through any special process but as noted above, it is better to use random text for security reasons.

Messages = <Messages-resource-name> The messages resource specifies where to deliver Director messages that are not associated with a specific Job. Most messages are specific to a job and will be directed to the Messages resource specified by the job. However, there are a few messages that can occur when no job is running. This directive is required.

Working Directory = <Directory> This directive is mandatory and specifies a directory in which the Director may put its status files. This directory should be used only by Bacula but may be shared by other Bacula daemons. However, please note, if this directory is shared with other Bacula daemons (the File daemon and Storage daemon), you must ensure that the **Name** given to each daemon is unique so that the temporary filenames used do not collide. By default the Bacula configure process creates unique daemon names by postfixing them with -dir, -fd, and -sd. Standard shell expansion of the **Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded. This directive is required. The working directory specified must already exist and be readable and writable by the Bacula daemon referencing it.

If you have specified a Director user and/or a Director group on your ./configure line with **--with-dir-user** and/or **--with-dir-group** the Working Directory owner and group will be set to those values.

Pid Directory = <Directory> This directive is mandatory and specifies a directory in which the Director may put its process Id file. The process Id file is used to shutdown Bacula and to prevent multiple copies of Bacula from running simultaneously. Standard shell expansion of the **Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

The PID directory specified must already exist and be readable and writable by the Bacula daemon referencing it

Typically on Linux systems, you will set this to: **/var/run**. If you are not installing Bacula in the system directories, you can use the **Working Directory** as defined above. This directive is required.

Scripts Directory = <Directory> This directive is optional and, if defined, specifies a directory in which the Director will look for the Python startup script **DirStartup.py**. This directory may be shared by other Bacula daemons. Standard shell expansion of the directory is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

QueryFile = <Path> This directive is mandatory and specifies a directory and file in which the Director can find the canned SQL statements for the **Query** command of the Console. Standard shell expansion of the **Path** is done when the configuration file is read so that values such as **$HOME** will be properly expanded. This directive is required.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Director to set a keepalive interval (heartbeat) in seconds on each of the sockets it opens for the Client resource. This value will override any specified at the Director level. It is implemented only on systems (Linux, ...) that provide the **setsockopt** TCP_KEEPIDLE function. The default value is zero, which means no change is made to the socket.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of total Director Jobs that should run concurrently. The default is set to 1, but you may set it to a larger number.

The Volume format becomes more complicated with multiple simultaneous jobs, consequently, restores may take longer if Bacula must sort through interleaved volume blocks from multiple simultaneous jobs. This can be avoided by having each simultaneous job write to a different volume or by using data spooling, which will first spool the data to disk simultaneously, then write one spool file at a time to the volume thus avoiding excessive interleaving of the different job blocks.

FD Connect Timeout = <time> where **time** is the time that the Director should continue attempting to contact the File daemon to start a job, and after which the Director will cancel the job. The default is 30 minutes.

SD Connect Timeout = <time> where **time** is the time that the Director should continue attempting to contact the Storage daemon to start a job, and after which the Director will cancel the job. The default is 30 minutes.

DirAddresses = <IP-address-specification> Specify the ports and addresses on which the Director daemon will listen for Bacula Console connections. Probably the simplest way to explain this is to show an example:

```
 DirAddresses  = {
    ip = { addr = 1.2.3.4; port = 1205;}
    ipv4 = {
        addr = 1.2.3.4; port = http;}
    ipv6 = {
        addr = 1.2.3.4;
        port = 1205;
    }
    ip = {
        addr = 1.2.3.4
        port = 1205
    }
    ip = { addr = 1.2.3.4 }
    ip = { addr = 201:220:222::2 }
    ip = {
        addr = bluedot.thun.net
    }
}
```

where ip, ip4, ip6, addr, and port are all keywords. Note, that the address can be specified as either a dotted quadruple, or IPv6 colon notation, or as a symbolic name (only in the ip specification). Also, port can be specified as a number or as the mnemonic value from the /etc/services file. If a port is not specified, the default will be used. If an ip section is specified, the resolution can be made either by IPv4 or IPv6. If ip4 is specified, then only IPv4 resolutions will be permitted, and likewise with ip6.

Please note that if you use the DirAddresses directive, you must not use either a DirPort or a DirAddress directive in the same resource.

DirPort = <port-number> Specify the port (a positive integer) on which the Director daemon will listen for Bacula Console connections. This same port number must be specified in the Director resource of the Console configuration file. The default is 9101, so normally this directive need not be specified. This directive should not be used if you specify DirAddresses (N.B plural) directive.

DirAddress = <IP-Address> This directive is optional, but if it is specified, it will cause the Director server (for the Console program) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple in string or quoted string format. If this directive is not specified, the Director will bind to any available address (the default). Note, unlike the DirAddresses specification noted above, this directive only permits a single address to be specified. This directive should not be used if you specify a DirAddresses (N.B. plural) directive.

DirSourceAddress = <IP-Address> This record is optional, and if it is specified, it will cause the Director server (when initiating connections to a storage or file daemon) to source its connections from the specified address. Only a single IP address may be specified. If this record is not specified, the Director server will source its outgoing connections according to the system routing table (the default).

Statistics Retention = <time> The `Statistics Retention` directive defines the length of time that Bacula will keep statistics job records in the Catalog database after the Job End time. (In `JobHistory` table) When this time period expires, and if user runs `prune stats` command, Bacula will prune (remove) Job records that are older than the specified period.

Theses statistics records aren't use for restore purpose, but mainly for capacity planning, billings, etc. See Statistics chapteror additional information.

See the Configuration chapter of this manual for additional details of time specification.

The default is 5 years.

VerId = <string> where <string> is an identifier which can be used for support purpose. This string is displayed using the `version` command.

MaxConsoleConnections = <number> where <number> is the maximum number of Console Connections that could run concurrently. The default is set to 20, but you may set it to a larger number.

The following is an example of a valid Director resource definition:

```
Director {
  Name = HeadMan
  WorkingDirectory = "$HOME/bacula/bin/working"
  Password = UA_password
  PidDirectory = "$HOME/bacula/bin/working"
  QueryFile = "$HOME/bacula/bin/query.sql"
  Messages = Standard
}
```

5.3 The Job Resource

The Job resource defines a Job (Backup, Restore, ...) that Bacula must perform. Each Job resource definition contains the name of a Client and a FileSet to backup, the Schedule for the Job, where the data are to be stored, and what media Pool can be used. In effect, each Job resource must specify What, Where, How, and When or FileSet, Storage, Backup/Restore/Level, and Schedule respectively. Note, the FileSet must be specified for a restore job for historical reasons, but it is no longer used.

Only a single type (**Backup**, **Restore**, ...) can be specified for any job. If you want to backup multiple FileSets on the same Client or multiple Clients, you must define a Job for each one.

Note, you define only a single Job to do the Full, Differential, and Incremental backups since the different backup levels are tied together by a unique Job name. Normally, you will have only one Job per Client, but if a client has a really huge number of files (more than several million), you might want to split it into to Jobs each with a different FileSet covering only part of the total files.

Multiple Storage daemons are not currently supported for Jobs, so if you do want to use multiple storage daemons, you will need to create a different Job and ensure that for each Job that the combination of Client and FileSet are unique. The Client and FileSet are what Bacula uses to restore a client, so if there are multiple Jobs with the same Client and FileSet or multiple Storage daemons that are used, the restore will not work. This problem can be resolved by defining multiple FileSet definitions (the names must be different, but the contents of the FileSets may be the same).

Job Start of the Job resource. At least one Job resource is required.

Name = <name> The Job name. This name can be specified on the **Run** command in the console program to start a job. If the name contains spaces, it must be specified between quotes. It is generally a good idea to give your job the same name as the Client that it will backup. This permits easy identification of jobs.

When the job actually runs, the unique Job Name will consist of the name you specify here followed by the date and time the job was scheduled for execution. This directive is required.

Enabled = <yes|no> This directive allows you to enable or disable automatic execution via the scheduler of a Job.

Type = <job-type> The **Type** directive specifies the Job type, which may be one of the following: **Backup**, **Restore**, **Verify**, or **Admin**. This directive is required. Within a particular Job Type, there are also Levels as discussed in the next item.

Backup Run a backup Job. Normally you will have at least one Backup job for each client you want to save. Normally, unless you turn off cataloging, most all the important statistics and data concerning files backed up will be placed in the catalog.

Restore Run a restore Job. Normally, you will specify only one Restore job which acts as a sort of prototype that you will modify using the console program in order to perform restores. Although certain basic information from a Restore job is saved in the catalog, it is very minimal compared to the information stored for a Backup job – for example, no File database entries are generated since no Files are saved.

Restore jobs cannot be automatically started by the scheduler as is the case for Backup, Verify and Admin jobs. To restore files, you must use the **restore** command in the console.

Verify Run a verify Job. In general, **verify** jobs permit you to compare the contents of the catalog to the file system, or to what was backed up. In addition, to verifying that a tape that was written can be read, you can also use **verify** as a sort of tripwire intrusion detection.

Admin Run an admin Job. An **Admin** job can be used to periodically run catalog pruning, if you do not want to do it at the end of each **Backup** Job. Although an Admin job is recorded in the catalog, very little data is saved.

Level = <job-level> The Level directive specifies the default Job level to be run. Each different Job Type (Backup, Restore, ...) has a different set of Levels that can be specified. The Level is normally overridden by a different value that is specified in the **Schedule** resource. This directive is not required, but must be specified either by a **Level** directive or as an override specified in the **Schedule** resource.

For a **Backup** Job, the Level may be one of the following:

Full When the Level is set to Full all files in the FileSet whether or not they have changed will be backed up.

Incremental When the Level is set to Incremental all files specified in the FileSet that have changed since the last successful backup of the the same Job using the same FileSet and Client, will be backed up. If the Director cannot find a previous valid Full backup then the job will be upgraded into a Full backup. When the Director looks for a valid backup record in the catalog database, it looks for a previous Job with:

- The same Job name.
- The same Client name.
- The same FileSet (any change to the definition of the FileSet such as adding or deleting a file in the Include or Exclude sections constitutes a different FileSet.
- The Job was a Full, Differential, or Incremental backup.
- The Job terminated normally (i.e. did not fail or was not canceled).
- The Job started no longer ago than **Max Full Interval**.

If all the above conditions do not hold, the Director will upgrade the Incremental to a Full save. Otherwise, the Incremental backup will be performed as requested.

The File daemon (Client) decides which files to backup for an Incremental backup by comparing start time of the prior Job (Full, Differential, or Incremental) against the time each file was last "modified" (st_mtime) and the time its attributes were last "changed"(st_ctime). If the file was modified or its attributes changed on or after this start time, it will then be backed up.

Some virus scanning software may change st_ctime while doing the scan. For example, if the virus scanning program attempts to reset the access time (st_atime), which Bacula does not use, it will cause st_ctime to change and hence Bacula will backup the file during an Incremental or Differential backup. In the case of Sophos virus scanning, you can prevent it from resetting the access time (st_atime) and hence changing st_ctime by using the **--no-reset-atime** option. For other software, please see their manual.

When Bacula does an Incremental backup, all modified files that are still on the system are backed up. However, any file that has been deleted since the last Full backup remains in the Bacula catalog, which means that if between a Full save and the time you do a restore, some files are deleted, those deleted files will also be restored. The deleted files will no longer appear in the catalog after doing another Full save.

In addition, if you move a directory rather than copy it, the files in it do not have their modification time (st_mtime) or their attribute change time (st_ctime) changed. As a consequence, those files will probably not be backed up by an Incremental or Differential backup which depend solely on these time stamps. If you move a directory, and wish it to be properly backed up, it is generally preferable to copy it, then delete the original.

However, to manage deleted files or directories changes in the catalog during an Incremental backup you can use `accurate` mode. This is quite memory consuming process. See Accurate mode for more details.

Differential When the Level is set to Differential all files specified in the FileSet that have changed since the last successful Full backup of the same Job will be backed up. If the Director cannot find a valid previous Full backup for the same Job, FileSet, and Client, backup, then the Differential job will be upgraded into a Full backup. When the Director looks for a valid Full backup record in the catalog database, it looks for a previous Job with:

- The same Job name.
- The same Client name.
- The same FileSet (any change to the definition of the FileSet such as adding or deleting a file in the Include or Exclude sections constitutes a different FileSet.
- The Job was a FULL backup.
- The Job terminated normally (i.e. did not fail or was not canceled).
- The Job started no longer ago than **Max Full Interval**.

If all the above conditions do not hold, the Director will upgrade the Differential to a Full save. Otherwise, the Differential backup will be performed as requested.

The File daemon (Client) decides which files to backup for a differential backup by comparing the start time of the prior Full backup Job against the time each file was last "modified" (st_mtime) and the time its attributes were last "changed" (st_ctime). If the file was modified or its attributes were changed on or after this start time, it will then be backed up. The start time used is displayed after the **Since** on the Job report. In rare cases, using the start time of the prior backup may cause some files to be backed up twice, but it ensures that no change is missed. As with the Incremental option, you should ensure that the clocks on your server and client are synchronized or as close as possible to avoid the possibility of a file being skipped. Note, on versions 1.33 or greater Bacula automatically makes the necessary adjustments to the time between the server and the client so that the times Bacula uses are synchronized.

When Bacula does a Differential backup, all modified files that are still on the system are backed up. However, any file that has been deleted since the last Full backup remains in the Bacula catalog, which means that if between a Full save and the time you do a restore, some files are deleted, those deleted files will also be restored. The deleted files will no longer appear in the catalog after doing another Full save. However, to remove deleted files from the catalog during a Differential backup is quite a time consuming process and not currently implemented in Bacula. It is, however, a planned future feature.

As noted above, if you move a directory rather than copy it, the files in it do not have their modification time (st_mtime) or their attribute change time (st_ctime) changed. As a consequence, those files will probably not be backed up by an Incremental or Differential backup which depend solely on these time stamps. If you move a directory, and wish it to be properly backed up, it is generally preferable to copy it, then delete the original. Alternatively, you can move the directory, then use the **touch** program to update the timestamps.

However, to manage deleted files or directories changes in the catalog during an Differential backup you can use `accurate` mode. This is quite memory consuming process. See Accurate mode for more details.

Every once and a while, someone asks why we need Differential backups as long as Incremental backups pickup all changed files. There are possibly many answers to this question, but the one that is the most important for me is that a Differential backup effectively merges all the Incremental and Differential backups since the last Full backup into a single Differential backup. This has two effects: 1. It gives some redundancy since the old backups could be used if the merged backup cannot be read. 2. More importantly, it reduces the number of Volumes that are needed to do a restore effectively eliminating the need to read all the volumes on which the preceding Incremental and Differential backups since the last Full are done.

For a **Restore** Job, no level needs to be specified.

For a **Verify** Job, the Level may be one of the following:

InitCatalog does a scan of the specified **FileSet** and stores the file attributes in the Catalog database. Since no file data is saved, you might ask why you would want to do this. It turns out to be a very simple and easy way to have a **Tripwire** like feature using **Bacula**. In other words, it allows you to save the state of a set of files defined by the **FileSet** and later check to see if those files have been modified or deleted and if any new files have been added. This can be used to detect system intrusion. Typically you would specify a **FileSet** that contains the set of system files that should not change (e.g. /sbin, /boot, /lib, /bin, ...). Normally, you run the **InitCatalog** level verify one time when your system is first setup, and then once again after each modification (upgrade) to your system. Thereafter, when your want to check the state of your system files, you use a **Verify level = Catalog**. This compares the results of your **InitCatalog** with the current state of the files.

Catalog Compares the current state of the files against the state previously saved during an **InitCatalog**. Any discrepancies are reported. The items reported are determined by the **verify** options specified on the **Include** directive in the specified **FileSet** (see the **FileSet** resource below for more details). Typically this command will be run once a day (or night) to check for any changes to your system files.

Please note! If you run two Verify Catalog jobs on the same client at the same time, the results will certainly be incorrect. This is because Verify Catalog modifies the Catalog database while running in order to track new files.

VolumeToCatalog This level causes Bacula to read the file attribute data written to the Volume from the last Job. The file attribute data are compared to the values saved in the Catalog database and any differences are reported. This is similar to the **Catalog** level except that instead of comparing the disk file attributes to the catalog database, the attribute data written to the Volume is read and compared to the catalog database. Although the attribute data including the signatures (MD5 or SHA1) are compared, the actual file data is not compared (it is not in the catalog).

Please note! If you run two Verify VolumeToCatalog jobs on the same client at the same time, the results will certainly be incorrect. This is because the Verify VolumeToCatalog modifies the Catalog database while running.

DiskToCatalog This level causes Bacula to read the files as they currently are on disk, and to compare the current file attributes with the attributes saved in the catalog from the last backup for the job specified on the **VerifyJob** directive. This level differs from the **Catalog** level described above by the fact that it doesn't compare against a previous Verify job but against a previous backup. When you run this level, you must supply the verify options on your Include statements. Those options determine what attribute fields are compared.

This command can be very useful if you have disk problems because it will compare the current state of your disk against the last successful backup, which may be several jobs.

Note, the current implementation (1.32c) does not identify files that have been deleted.

Accurate = <yes|no> In accurate mode, the File daemon knowns exactly which files were present after the last backup. So it is able to handle deleted or renamed files.

When restoring a FileSet for a specified date (including "most recent"), Bacula is able to restore exactly the files and directories that existed at the time of the last backup prior to that date including ensuring that deleted files are actually deleted, and renamed directories are restored properly.

In this mode, the File daemon must keep data concerning all files in memory. So you do not have sufficient memory, the restore may either be terribly slow or fail.

For 500.000 files (a typical desktop linux system), it will require approximately 64 Megabytes of RAM on your File daemon to hold the required information.

Verify Job = <Job-Resource-Name> If you run a verify job without this directive, the last job run will be compared with the catalog, which means that you must immediately follow a backup by a verify command. If you specify a **Verify Job** Bacula will find the last job with that name that ran. This permits you to run all your backups, then run Verify jobs on those that you wish to be verified (most often a **VolumeToCatalog**) so that the tape just written is re-read.

JobDefs = <JobDefs-Resource-Name> If a JobDefs-Resource-Name is specified, all the values contained in the named JobDefs resource will be used as the defaults for the current Job. Any value that you explicitly define in the current Job resource, will override any defaults specified in the JobDefs resource. The use of this directive permits writing much more compact Job resources where the bulk of the directives are defined in one or more JobDefs. This is particularly useful if you have many similar Jobs but with minor variations such as different Clients. A simple example of the use of JobDefs is provided in the default bacula-dir.conf file.

Bootstrap = <bootstrap-file> The Bootstrap directive specifies a bootstrap file that, if provided, will be used during **Restore** Jobs and is ignored in other Job types. The **bootstrap** file contains the list of tapes to be used in a restore Job as well as which files are to be restored. Specification of this directive is optional, and if specified, it is used only for a restore job. In addition, when running a Restore job from the console, this value can be changed.

If you use the **Restore** command in the Console program, to start a restore job, the **bootstrap** file will be created automatically from the files you select to be restored.

For additional details of the **bootstrap** file, please see Restoring Files with the Bootstrap File chapter of this manual.

Write Bootstrap = <bootstrap-file-specification> The **writebootstrap** directive specifies a file name where Bacula will write a **bootstrap** file for each Backup job run. This directive applies only to Backup Jobs. If the Backup job is a Full save, Bacula will erase any current contents of the specified file before writing the bootstrap records. If the Job is an Incremental or Differential save, Bacula will append the current bootstrap record to the end of the file.

Using this feature, permits you to constantly have a bootstrap file that can recover the current state of your system. Normally, the file specified should be a mounted drive on another machine, so that if your hard disk is lost, you will immediately have a bootstrap record available. Alternatively, you should copy the bootstrap file to another machine after it is updated. Note, it is a good idea to write a separate bootstrap file for each Job backed up including the job that backs up your catalog database.

If the **bootstrap-file-specification** begins with a vertical bar (—), Bacula will use the specification as the name of a program to which it will pipe the bootstrap record. It could for example be a shell script that emails you the bootstrap record.

On versions 1.39.22 or greater, before opening the file or executing the specified command, Bacula performs character substitution like in RunScript directive. To automatically manage your bootstrap files, you can use this in your **JobDefs** resources:

```
JobDefs {
   Write Bootstrap = "%c_%n.bsr"
   ...
}
```

For more details on using this file, please see the chapter entitled The Bootstrap File of this manual.

Client = <client-resource-name> The Client directive specifies the Client (File daemon) that will be used in the current Job. Only a single Client may be specified in any one Job. The Client runs on the machine to be backed up, and sends the requested files to the Storage daemon for backup, or receives them when restoring. For additional details, see the Client Resource section of this chapter. This directive is required.

FileSet = <FileSet-resource-name> The FileSet directive specifies the FileSet that will be used in the current Job. The FileSet specifies which directories (or files) are to be backed up, and what options to use (e.g. compression, ...). Only a single FileSet resource may be specified in any one Job. For additional details, see the FileSet Resource section of this chapter. This directive is required.

Messages = <messages-resource-name> The Messages directive defines what Messages resource should be used for this job, and thus how and where the various messages are to be delivered. For example, you can direct some messages to a log file, and others can be sent by email. For additional details, see the Messages Resource Chapter of this manual. This directive is required.

Pool = <pool-resource-name> The Pool directive defines the pool of Volumes where your data can be backed up. Many Bacula installations will use only the **Default** pool. However, if you want to specify a different set of Volumes for different Clients or different Jobs, you will probably want to use Pools. For additional details, see the Pool Resource section of this chapter. This directive is required.

Full Backup Pool = <pool-resource-name> The *Full Backup Pool* specifies a Pool to be used for Full backups. It will override any Pool specification during a Full backup. This directive is optional.

Differential Backup Pool = <pool-resource-name> The *Differential Backup Pool* specifies a Pool to be used for Differential backups. It will override any Pool specification during a Differential backup. This directive is optional.

Incremental Backup Pool = <pool-resource-name> The *Incremental Backup Pool* specifies a Pool to be used for Incremental backups. It will override any Pool specification during an Incremental backup. This directive is optional.

Schedule = <schedule-name> The Schedule directive defines what schedule is to be used for the Job. The schedule in turn determines when the Job will be automatically started and what Job level (i.e. Full, Incremental, ...) is to be run. This directive is optional, and if left out, the Job can only be started manually using the Console program. Although you may specify only a single Schedule resource for any one job, the Schedule resource may contain multiple **Run** directives, which allow you to run the Job at many different times, and each **run** directive permits overriding the default Job Level Pool, Storage, and Messages resources. This gives considerable flexibility in what can be done with a single Job. For additional details, see the Schedule Resource Chapter of this manual.

Storage = <storage-resource-name> The Storage directive defines the name of the storage services where you want to backup the FileSet data. For additional details, see the Storage Resource Chapter of this manual. The Storage resource may also be specified in the Job's Pool resource, in which case the value in the Pool resource overrides any value in the Job. This Storage resource definition is not required by either the Job resource or in the Pool, but it must be specified in one or the other, if not an error will result.

Max Start Delay = <time> The time specifies the maximum delay between the scheduled time and the actual start time for the Job. For example, a job can be scheduled to run at 1:00am, but because other jobs are running, it may wait to run. If the delay is set to 3600 (one hour) and the job has not begun to run by 2:00am, the job will be canceled. This can be useful, for example, to prevent jobs from running during day time hours. The default is 0 which indicates no limit.

Max Run Time = <time> The time specifies the maximum allowed time that a job may run, counted from when the job starts, (**not** necessarily the same as when the job was scheduled).

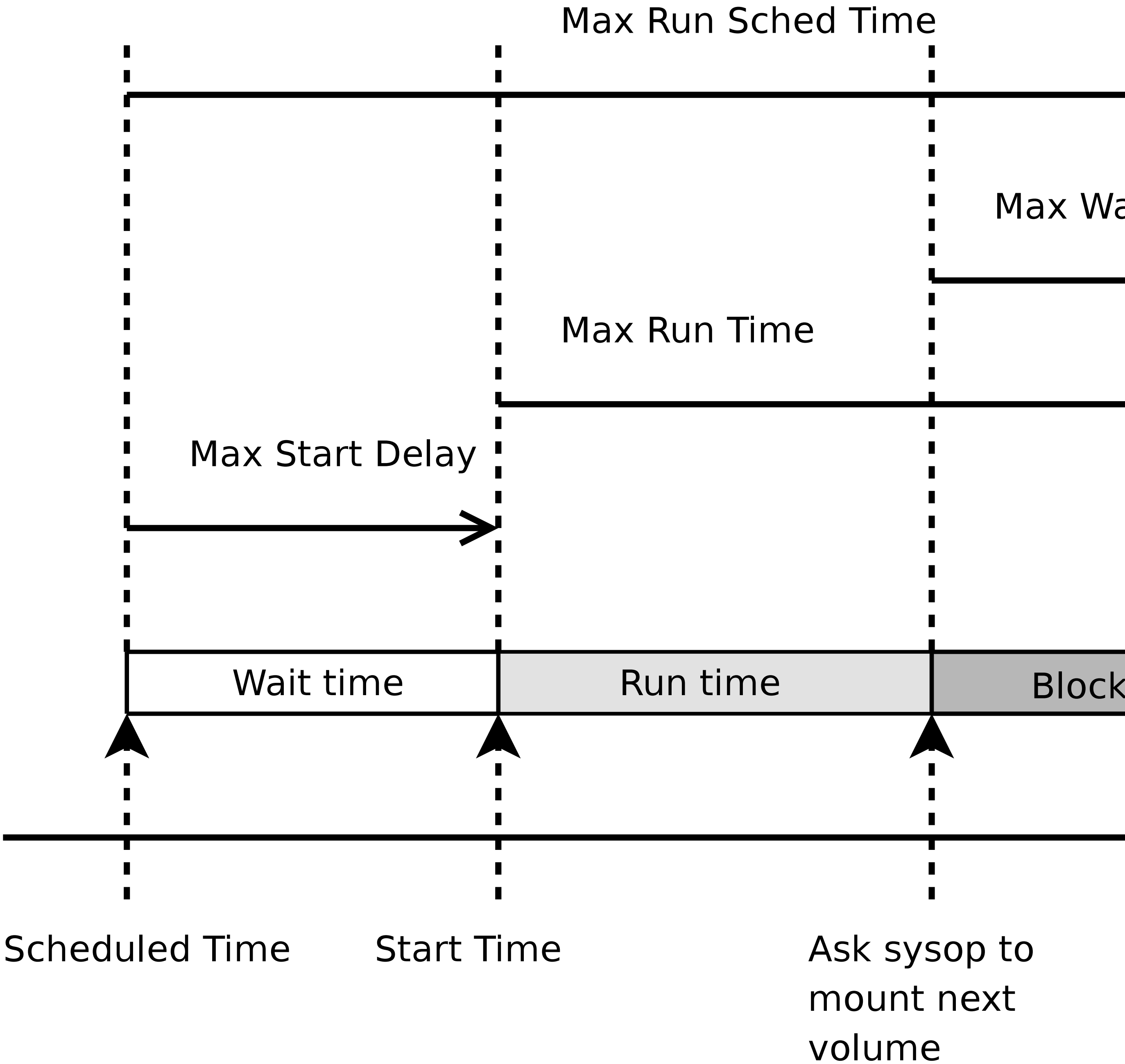
Incremental—Differential Max Wait Time = <time> Theses directives have been deprecated in favor of `Incremental|Differential Max Run Time` since bacula 2.3.18.

Incremental Max Run Time = <time> The time specifies the maximum allowed time that an Incremental backup job may run, counted from when the job starts, (**not** necessarily the same as when the job was scheduled).

Differential Max Wait Time = <time> The time specifies the maximum allowed time that a Differential backup job may run, counted from when the job starts, (**not** necessarily the same as when the job was scheduled).

Max Run Sched Time = <time> The time specifies the maximum allowed time that a job may run, counted from when the job was scheduled. This can be useful to prevent jobs from running during working hours. We can see it like `Max Start Delay + Max Run Time`.

Max Wait Time = <time> The time specifies the maximum allowed time that a job may block waiting for a resource (such as waiting for a tape to be mounted, or waiting for the storage or file daemons to perform their duties), counted from the when the job starts, (**not** necessarily the same as when the job was scheduled). This directive works as expected since bacula 2.3.18.



Max Full Interval = <time> The time specifies the maximum allowed age (counting from start time) of the most recent successful Full backup that is required in order to run Incremental or Differential backup jobs. If the most recent Full backup is older than this interval, Incremental and Differential backups will be upgraded to Full backups automatically. If this directive is not present, or specified as 0, then the age of the previous Full backup is not considered.

Prefer Mounted Volumes = <yes|no> If the Prefer Mounted Volumes directive is set to **yes** (default yes), the Storage daemon is requested to select either an Autochanger or a drive with a valid Volume already mounted in preference to a drive that is not ready. This means that all jobs will attempt to append to the same Volume (providing the Volume is appropriate – right Pool, ... for that job), unless you are using multiple pools. If no drive with a suitable Volume is available, it will select the first available drive. Note, any Volume that has been requested to be mounted, will be considered valid as a mounted volume by another job. This if multiple jobs start at the same time and they all prefer mounted volumes, the first job will request the mount, and the other jobs will use the same volume.

If the directive is set to **no**, the Storage daemon will prefer finding an unused drive, otherwise, each job started will append to the same Volume (assuming the Pool is the same for all jobs). Setting Prefer Mounted Volumes to no can be useful for those sites with multiple drive autochangers that prefer to maximize backup throughput at the expense of using additional drives and Volumes. This means that the job will prefer to use an unused drive rather than use a drive that is already in use.

Despite the above, we recommend against setting this directive to **no** since it tends to add a lot of swapping of Volumes between the different drives and can easily lead to deadlock situations in the Storage daemon. We will accept bug reports against it, but we cannot guarantee that we will be able to fix the problem in a reasonable time.

A better alternative for using multiple drives is to use multiple pools so that Bacula will be forced to mount Volumes from those Pools on different drives.

Prune Jobs = <yes|no> Normally, pruning of Jobs from the Catalog is specified on a Client by Client basis in the Client resource with the **AutoPrune** directive. If this directive is specified (not normally) and the value is **yes**, it will override the value specified in the Client resource. The default is **no**.

Prune Files = <yes|no> Normally, pruning of Files from the Catalog is specified on a Client by Client basis in the Client resource with the **AutoPrune** directive. If this directive is specified (not normally) and the value is **yes**, it will override the value specified in the Client resource. The default is **no**.

Prune Volumes = <yes|no> Normally, pruning of Volumes from the Catalog is specified on a Client by Client basis in the Client resource with the **AutoPrune** directive. If this directive is specified (not normally) and the value is **yes**, it will override the value specified in the Client resource. The default is **no**.

RunScript {<body-of-runscript>} The RunScript directive behaves like a resource in that it requires opening and closing braces around a number of directives that make up the body of the runscript.

The specified **Command** (see below for details) is run as an external program prior or after the current Job. This is optional. By default, the program is executed on the Client side like in `ClientRunXXXJob`.

Console options are special commands that are sent to the director instead of the OS. At this time, console command ouputs are redirected to log with the jobid 0.

You can use following console command : `delete`, `disable`, `enable`, `estimate`, `list`, `llist`, `memory`, `prune`, `purge`, `reload`, `status`, `setdebug`, `show`, `time`, `trace`, `update`, `version`, `.client`, `.jobs`, `.pool`, `.storage`. See console chapter for more information. You need to specify needed information on command line, nothing will be prompted. Example :

```
Console = "prune files client=%c"
Console = "update stats age=3"
```

You can specify more than one Command/Console option per RunScript.

You can use following options may be specified in the body of the runscript:

Options	Value	Default	Information
Runs On Success	Yes/No	*Yes*	Run command if JobStatus is successful
Runs On Failure	Yes/No	*No*	Run command if JobStatus isn't successful
Runs On Client	Yes/No	*Yes*	Run command on client
Runs When	Before—After—Always—*AfterVSS*	*Never*	When run commands
Fail Job On Error	Yes/No	*Yes*	Fail job if script returns something different fro
Command			Path to your script
Console			Console command

Any output sent by the command to standard output will be included in the Bacula job report. The command string must be a valid program name or name of a shell script.

In addition, the command string is parsed then fed to the OS, which means that the path will be searched to execute your specified command, but there is no shell interpretation, as a consequence, if you invoke complicated commands or want any shell features such as redirection or piping, you must call a shell script and do it inside that script.

Before submitting the specified command to the operating system, Bacula performs character substitution of the following characters:

```
%% = %
%c = Client's name
%d = Director's name
%e = Job Exit Status
%i = JobId
%j = Unique Job id
%l = Job Level
%n = Job name
%s = Since time
%t = Job type (Backup, ...)
%v = Volume name (Only on director side)
```

The Job Exit Status code %e edits the following values:

- OK
- Error
- Fatal Error
- Canceled
- Differences
- Unknown term code

Thus if you edit it on a command line, you will need to enclose it within some sort of quotes.

You can use these following shortcuts:

Keyword	RunsOnSuccess	RunsOnFailure	FailJobOnError	Runs On Client	RunsWhen
Run Before Job			Yes	No	Before
Run After Job	Yes	No		No	After
Run After Failed Job	No	Yes		No	After
Client Run Before Job			Yes	Yes	Before
Client Run After Job	Yes	No		Yes	After

Examples:

```
RunScript {
    RunsWhen = Before
    FailJobOnError = No
    Command = "/etc/init.d/apache stop"
}

RunScript {
    RunsWhen = After
    RunsOnFailure = yes
    Command = "/etc/init.d/apache start"
}
```

Notes about ClientRunBeforeJob

For compatibility reasons, with this shortcut, the command is executed directly when the client recieve it. And if the command is in error, other remote runscripts will be discarded. To be sure that all commands will be sent and executed, you have to use RunScript syntax.

Special Windows Considerations

You can run scripts just after snapshots initializations with *AfterVSS* keyword.

In addition, for a Windows client on version 1.33 and above, please take note that you must ensure a correct path to your script. The script or program can be a .com, .exe or a .bat file. If you just put the program name in then Bacula will search using the same rules that cmd.exe uses (current directory, Bacula bin directory, and PATH). It will even try the different extensions in the same order as cmd.exe. The command can be anything that cmd.exe or command.com will recognize as an executable file.

However, if you have slashes in the program name then Bacula figures you are fully specifying the name, so you must also explicitly add the three character extension.

The command is run in a Win32 environment, so Unix like commands will not work unless you have installed and properly configured Cygwin in addition to and separately from Bacula.

The System %Path% will be searched for the command. (under the environment variable dialog you have have both System Environment and User Environment, we believe that only the System environment will be available to bacula-fd, if it is running as a service.)

System environment variables can be referenced with %var% and used as either part of the command name or arguments.

So if you have a script in the Bacula
bin directory then the following lines should work fine:

```
        Client Run Before Job = systemstate
or
        Client Run Before Job = systemstate.bat
or
        Client Run Before Job = "systemstate"
or
        Client Run Before Job = "systemstate.bat"
or
        ClientRunBeforeJob = "\"C:/Program Files/Bacula/systemstate.bat\""
```

The outer set of quotes is removed when the configuration file is parsed. You need to escape the inner quotes so that they are there when the code that parses the command line for execution runs so it can tell what the program name is.

```
ClientRunBeforeJob = "\"C:/Program Files/Software
     Vendor/Executable\" /arg1 /arg2 \"foo bar\""
```

The special characters

```
&<>()@^|
```

will need to be quoted, if they are part of a filename or argument.

If someone is logged in, a blank "command" window running the commands will be present during the execution of the command.

Some Suggestions from Phil Stracchino for running on Win32 machines with the native Win32 File daemon:

1. You might want the ClientRunBeforeJob directive to specify a .bat file which runs the actual client-side commands, rather than trying to run (for example) regedit /e directly.
2. The batch file should explicitly 'exit 0' on successful completion.
3. The path to the batch file should be specified in Unix form:
   ClientRunBeforeJob = "c:/bacula/bin/systemstate.bat"
   rather than DOS/Windows form:
   ClientRunBeforeJob =
   "c:\bacula\bin\systemstate.bat" INCORRECT

For Win32, please note that there are certain limitations:

ClientRunBeforeJob = "C:/Program Files/Bacula/bin/pre-exec.bat"

Lines like the above do not work because there are limitations of cmd.exe that is used to execute the command. Bacula prefixes the string you supply with **cmd.exe /c** . To test that your command works you should type **cmd /c "C:/Program Files/test.exe"** at a cmd prompt and see what happens. Once the command is correct insert a backslash (\) before each double quote ("), and then put quotes around the whole thing when putting it in the director's .conf file. You either need to have only one set of quotes or else use the short name and don't put quotes around the command path.

Below is the output from cmd's help as it relates to the command line passed to the /c option.

If /C or /K is specified, then the remainder of the command line after the switch is processed as a command line, where the following logic is used to process quote (") characters:

1. If all of the following conditions are met, then quote characters on the command line are preserved:

- no /S switch.
- exactly two quote characters.
- no special characters between the two quote characters, where special is one of: `&<>()@^|`
- there are one or more whitespace characters between the the two quote characters.
- the string between the two quote characters is the name of an executable file.

2. Otherwise, old behavior is to see if the first character is a quote character and if so, strip the leading character and remove the last quote character on the command line, preserving any text after the last quote character.

The following example of the use of the Client Run Before Job directive was submitted by a user:
You could write a shell script to back up a DB2 database to a FIFO. The shell script is:

```
 #!/bin/sh
 # ===== backupdb.sh
 DIR=/u01/mercuryd

 mkfifo $DIR/dbpipe
 db2 BACKUP DATABASE mercuryd TO $DIR/dbpipe WITHOUT PROMPTING &
 sleep 1
```

The following line in the Job resource in the bacula-dir.conf file:

```
 Client Run Before Job = "su - mercuryd -c \"/u01/mercuryd/backupdb.sh '%t'
'%l'\""
```

When the job is run, you will get messages from the output of the script stating that the backup has started. Even though the command being run is backgrounded with &, the job will block until the "db2 BACKUP DATABASE" command, thus the backup stalls.

To remedy this situation, the "db2 BACKUP DATABASE" line should be changed to the following:

```
 db2 BACKUP DATABASE mercuryd TO $DIR/dbpipe WITHOUT PROMPTING > $DIR/backup.log
2>&1 < /dev/null &
```

It is important to redirect the input and outputs of a backgrounded command to /dev/null to prevent the script from blocking.

Run Before Job = <command> The specified **command** is run as an external program prior to running the current Job. This directive is not required, but if it is defined, and if the exit code of the program run is non-zero, the current Bacula job will be canceled.

```
Run Before Job = "echo test"
```

it's equivalent to :

```
RunScript {
 Command = "echo test"
 RunsOnClient = No
 RunsWhen = Before
}
```

Lutz Kittler has pointed out that using the RunBeforeJob directive can be a simple way to modify your schedules during a holiday. For example, suppose that you normally do Full backups on Fridays, but Thursday and Friday are holidays. To avoid having to change tapes between Thursday and Friday when no one is in the office, you can create a RunBeforeJob that returns a non-zero status on Thursday and zero on all other days. That way, the Thursday job will not run, and on Friday the tape you inserted on Wednesday before leaving will be used.

Run After Job = <command> The specified **command** is run as an external program if the current job terminates normally (without error or without being canceled). This directive is not required. If the exit code of the program run is non-zero, Bacula will print a warning message. Before submitting the specified command to the operating system, Bacula performs character substitution as described above for the **RunScript** directive.

An example of the use of this directive is given in the Tips Chapter of this manual.

See the **Run After Failed Job** if you want to run a script after the job has terminated with any non-normal status.

Run After Failed Job = <command> The specified **command** is run as an external program after the current job terminates with any error status. This directive is not required. The command string must be a valid program name or name of a shell script. If the exit code of the program run is non-zero, Bacula will print a warning message. Before submitting the specified command to the operating system, Bacula performs character substitution as described above for the **RunScript** directive. Note, if you wish that your script will run regardless of the exit status of the Job, you can use this :

```
RunScript {
 Command = "echo test"
 RunsWhen = After
 RunsOnFailure = yes
 RunsOnClient  = no
 RunsOnSuccess = yes    # default, you can drop this line
}
```

An example of the use of this directive is given in the Tips Chapter of this manual.

Client Run Before Job = <command> This directive is the same as **Run Before Job** except that the program is run on the client machine. The same restrictions apply to Unix systems as noted above for the **RunScript**.

Client Run After Job = <command> The specified **command** is run on the client machine as soon as data spooling is complete in order to allow restarting applications on the client as soon as possible. .

Note, please see the notes above in **RunScript** concerning Windows clients.

Rerun Failed Levels = <yes|no> If this directive is set to **yes** (default no), and Bacula detects that a previous job at a higher level (i.e. Full or Differential) has failed, the current job level will be upgraded to the higher level. This is particularly useful for Laptops where they may often be unreachable, and if a prior Full save has failed, you wish the very next backup to be a Full save rather than whatever level it is started as.

There are several points that must be taken into account when using this directive: first, a failed job is defined as one that has not terminated normally, which includes any running job of the same name (you need to ensure that two jobs of the same name do not run simultaneously); secondly, the **Ignore FileSet Changes** directive is not considered when checking for failed levels, which means that any FileSet change will trigger a rerun.

Spool Data = <yes|no> If this directive is set to **yes** (default no), the Storage daemon will be requested to spool the data for this Job to disk rather than write it directly to tape. Once all the data arrives or the spool files' maximum sizes are reached, the data will be despooled and written to tape. Spooling data prevents tape shoe-shine (start and stop) during Incremental saves. If you are writing to a disk file using this option will probably just slow down the backup jobs.

NOTE: When this directive is set to yes, Spool Attributes is also automatically set to yes.

Spool Attributes = <yes|no> The default is set to **no**, which means that the File attributes are sent by the Storage daemon to the Director as they are stored on tape. However, if you want to avoid the possibility that database updates will slow down writing to the tape, you may want to set the value to **yes**, in which case the Storage daemon will buffer the File attributes and Storage coordinates to a temporary file in the Working Directory, then when writing the Job data to the tape is completed, the attributes and storage coordinates will be sent to the Director.

NOTE: When Spool Data is set to yes, Spool Attributes is also automatically set to yes.

Where = <directory> This directive applies only to a Restore job and specifies a prefix to the directory name of all files being restored. This permits files to be restored in a different location from which they were saved. If **Where** is not specified or is set to backslash (/), the files will be restored to their original location. By default, we have set **Where** in the example configuration files to be **/tmp/bacula-restores**. This is to prevent accidental overwriting of your files.

Add Prefix = <directory> This directive applies only to a Restore job and specifies a prefix to the directory name of all files being restored. This will use File Relocation feature implemented in Bacula 2.1.8 or later.

Add Suffix = <extention> This directive applies only to a Restore job and specifies a suffix to all files being restored. This will use File Relocation feature implemented in Bacula 2.1.8 or later.

Using `Add Suffix=.old`, `/etc/passwd` will be restored to `/etc/passwsd.old`

Strip Prefix = <directory> This directive applies only to a Restore job and specifies a prefix to remove from the directory name of all files being restored. This will use the File Relocation feature implemented in Bacula 2.1.8 or later.

Using `Strip Prefix=/etc`, `/etc/passwd` will be restored to `/passwd`

Under Windows, if you want to restore `c:/files` to `d:/files`, you can use :

```
Strip Prefix = c:
Add Prefix = d:
```

RegexWhere = <expressions> This directive applies only to a Restore job and specifies a regex filename manipulation of all files being restored. This will use File Relocation feature implemented in Bacula 2.1.8 or later.

For more informations about how use this option, see this.

Replace = <replace-option> This directive applies only to a Restore job and specifies what happens when Bacula wants to restore a file or directory that already exists. You have the following options for **replace-option**:

always when the file to be restored already exists, it is deleted and then replaced by the copy that was backed up. This is the default value.

ifnewer if the backed up file (on tape) is newer than the existing file, the existing file is deleted and replaced by the back up.

ifolder if the backed up file (on tape) is older than the existing file, the existing file is deleted and replaced by the back up.

never if the backed up file already exists, Bacula skips restoring this file.

Prefix Links=<yes|no> If a **Where** path prefix is specified for a recovery job, apply it to absolute links as well. The default is **No**. When set to **Yes** then while restoring files to an alternate directory, any absolute soft links will also be modified to point to the new alternate directory. Normally this is what is desired – i.e. everything is self consistent. However, if you wish to later move the files to their original locations, all files linked with absolute names will be broken.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs from the current Job resource that can run concurrently. Note, this directive limits only Jobs with the same name as the resource in which it appears. Any other restrictions on the maximum concurrent jobs such as in the Director, Client, or Storage resources will also apply in addition to the limit specified here. The default is set to 1, but you may set it to a larger number. We strongly recommend that you read the WARNING documented under Maximum Concurrent Jobs in the Director's resource.

Reschedule On Error = <yes|no> If this directive is enabled, and the job terminates in error, the job will be rescheduled as determined by the **Reschedule Interval** and **Reschedule Times** directives. If you cancel the job, it will not be rescheduled. The default is **no** (i.e. the job will not be rescheduled).

This specification can be useful for portables, laptops, or other machines that are not always connected to the network or switched on.

Reschedule Interval = <time-specification> If you have specified **Reschedule On Error = yes** and the job terminates in error, it will be rescheduled after the interval of time specified by **time-specification**. See the time specification formats in the Configure chapter for details of time specifications. If no interval is specified, the job will not be rescheduled on error.

Reschedule Times = <count> This directive specifies the maximum number of times to reschedule the job. If it is set to zero (the default) the job will be rescheduled an indefinite number of times.

Run = <job-name> The Run directive (not to be confused with the Run option in a Schedule) allows you to start other jobs or to clone jobs. By using the cloning keywords (see below), you can backup the same data (or almost the same data) to two or more drives at the same time. The **job-name** is normally the same name as the current Job resource (thus creating a clone). However, it may be any Job name, so one job may start other related jobs.

The part after the equal sign must be enclosed in double quotes, and can contain any string or set of options (overrides) that you can specify when entering the Run command from the console. For example **storage=DDS-4 ...**. In addition, there are two special keywords that permit you to clone the current job. They are **level=%l** and **since=%s**. The %l in the level keyword permits entering the actual level of the current job and the %s in the since keyword permits putting the same time for comparison as used on the current job. Note, in the case of the since keyword, the %s must be enclosed in double quotes, and thus they must be preceded by a backslash since they are already inside quotes. For example:

```
run = "Nightly-backup level=%l since=\"%s\" storage=DDS-4"
```

A cloned job will not start additional clones, so it is not possible to recurse.

Please note that all cloned jobs, as specified in the Run directives are submitted for running before the original job is run (while it is being initialized). This means that any clone job will actually start before the original job, and may even block the original job from starting until the original job finishes unless you allow multiple simultaneous jobs. Even if you set a lower priority on the clone job, if no other jobs are running, it will start before the original job.

If you are trying to prioritize jobs by using the clone feature (Run directive), you will find it much easier to do using a RunScript resource, or a RunBeforeJob directive.

Priority = <number> This directive permits you to control the order in which your jobs will be run by specifying a positive non-zero number. The higher the number, the lower the job priority. Assuming you are not running concurrent jobs, all queued jobs of priority 1 will run before queued jobs of priority 2 and so on, regardless of the original scheduling order.

The priority only affects waiting jobs that are queued to run, not jobs that are already running. If one or more jobs of priority 2 are already running, and a new job is scheduled with priority 1, the currently running priority 2 jobs must complete before the priority 1 job is run, unless Allow Mixed Priority is set.

The default priority is 10.

If you want to run concurrent jobs you should keep these points in mind:

- See Running Concurrent Jobs on how to setup concurrent jobs.
- Bacula concurrently runs jobs of only one priority at a time. It will not simultaneously run a priority 1 and a priority 2 job.
- If Bacula is running a priority 2 job and a new priority 1 job is scheduled, it will wait until the running priority 2 job terminates even if the Maximum Concurrent Jobs settings would otherwise allow two jobs to run simultaneously.
- Suppose that bacula is running a priority 2 job and a new priority 1 job is scheduled and queued waiting for the running priority 2 job to terminate. If you then start a second priority 2 job, the waiting priority 1 job will prevent the new priority 2 job from running concurrently with the running priority 2 job. That is: as long as there is a higher priority job waiting to run, no new lower priority jobs will start even if the Maximum Concurrent Jobs settings would normally allow them to run. This ensures that higher priority jobs will be run as soon as possible.

If you have several jobs of different priority, it may not best to start them at exactly the same time, because Bacula must examine them one at a time. If by Bacula starts a lower priority job first, then it will run before your high priority jobs. If you experience this problem, you may avoid it by starting any higher priority jobs a few seconds before lower priority ones. This insures that Bacula will examine the jobs in the correct order, and that your priority scheme will be respected.

Allow Mixed Priority = <yes|no> This directive is only implemented in version 2.5 and later. When set to **yes** (default **no**), this job may run even if lower priority jobs are already running. This means a high priority job will not have to wait for other jobs to finish before starting. The scheduler will only mix priorities when all running jobs have this set to true.

Note that only higher priority jobs will start early. Suppose the director will allow two concurrent jobs, and that two jobs with priority 10 are running, with two more in the queue. If a job with priority 5 is added to the queue, it will be run as soon as one of the running jobs finishes. However, new priority 10 jobs will not be run until the priority 5 job has finished.

Write Part After Job = <yes|no> This directive is only implemented in version 1.37 and later. If this directive is set to **yes** (default **no**), a new part file will be created after the job is finished.

It should be set to **yes** when writing to devices that require mount (for example DVD), so you are sure that the current part, containing this job's data, is written to the device, and that no data is left in the temporary file on the hard disk. However, on some media, like DVD+R and DVD-R, a lot of space (about 10Mb) is lost every time a part is written. So, if you run several jobs each after another, you could set this directive to **no** for all jobs, except the last one, to avoid wasting too much space, but to ensure that the data is written to the medium when all jobs are finished.

This directive is ignored with tape and FIFO devices.

The following is an example of a valid Job resource definition:

```
Job {
  Name = "Minou"
  Type = Backup
  Level = Incremental                 # default
  Client = Minou
  FileSet="Minou Full Set"
  Storage = DLTDrive
  Pool = Default
  Schedule = "MinouWeeklyCycle"
  Messages = Standard
}
```

5.4 The JobDefs Resource

The JobDefs resource permits all the same directives that can appear in a Job resource. However, a JobDefs resource does not create a Job, rather it can be referenced within a Job to provide defaults for that Job. This permits you to concisely define several nearly identical Jobs, each one referencing a JobDefs resource which contains the defaults. Only the changes from the defaults need to be mentioned in each Job.

5.5 The Schedule Resource

The Schedule resource provides a means of automatically scheduling a Job as well as the ability to override the default Level, Pool, Storage and Messages resources. If a Schedule resource is not referenced in a Job, the Job can only be run manually. In general, you specify an action to be taken and when.

Schedule Start of the Schedule directives. No **Schedule** resource is required, but you will need at least one if you want Jobs to be automatically started.

Name = <name> The name of the schedule being defined. The Name directive is required.

Run = <Job-overrides> <Date-time-specification> The Run directive defines when a Job is to be run, and what overrides if any to apply. You may specify multiple **run** directives within a **Schedule** resource. If you do, they will all be applied (i.e. multiple schedules). If you have two **Run** directives that start at the same time, two Jobs will start at the same time (well, within one second of each other).

The **Job-overrides** permit overriding the Level, the Storage, the Messages, and the Pool specifications provided in the Job resource. In addition, the FullPool, the IncrementalPool, and the DifferentialPool specifications permit overriding the Pool specification according to what backup Job Level is in effect.

By the use of overrides, you may customize a particular Job. For example, you may specify a Messages override for your Incremental backups that outputs messages to a log file, but for your weekly or monthly Full backups, you may send the output by email by using a different Messages override.

Job-overrides are specified as: **keyword=value** where the keyword is Level, Storage, Messages, Pool, FullPool, DifferentialPool, or IncrementalPool, and the **value** is as defined on the respective directive formats for the Job resource. You may specify multiple **Job-overrides** on one **Run** directive by separating them with one or more spaces or by separating them with a trailing comma. For example:

Level=Full is all files in the FileSet whether or not they have changed.

Level=Incremental is all files that have changed since the last backup.

Pool=Weekly specifies to use the Pool named **Weekly**.

Storage=DLT_Drive specifies to use **DLT_Drive** for the storage device.

Messages=Verbose specifies to use the **Verbose** message resource for the Job.

FullPool=Full specifies to use the Pool named **Full** if the job is a full backup, or is upgraded from another type to a full backup.

DifferentialPool=Differential specifies to use the Pool named **Differential** if the job is a differential backup.

IncrementalPool=Incremental specifies to use the Pool named **Incremental** if the job is an incremental backup.

SpoolData=yes|no tells Bacula to request the Storage daemon to spool data to a disk file before writing it to the Volume (normally a tape). Thus the data is written in large blocks to the Volume rather than small blocks. This directive is particularly useful when running multiple simultaneous backups to tape. It prevents interleaving of the job data and reduces or eliminates tape drive stop and start commonly known as "shoe-shine".

SpoolSize=*bytes* where the bytes specify the maximum spool size for this job. The default is take from Device Maximum Spool Size limit. This directive is available only in Bacula version 2.3.5 or later.

WritePartAfterJob=yes|no tells Bacula to request the Storage daemon to write the current part file to the device when the job is finished (see Write Part After Job directive in the Job resource). Please note, this directive is implemented only in version 1.37 and later. The default is yes. We strongly recommend that you keep this set to yes otherwise, when the last job has finished one part will remain in the spool file and restore may or may not work.

Date-time-specification determines when the Job is to be run. The specification is a repetition, and as a default Bacula is set to run a job at the beginning of the hour of every hour of every day of every week of every month of every year. This is not normally what you want, so you must specify or limit when you want the job to run. Any specification given is assumed to be repetitive in nature and will serve to override or limit the default repetition. This is done by specifying masks or times for the hour, day of the month, day of the week, week of the month, week of the year, and month when you want the job to run. By specifying one or more of the above, you can define a schedule to repeat at almost any frequency you want.

Basically, you must supply a **month**, **day**, **hour**, and **minute** the Job is to be run. Of these four items to be specified, **day** is special in that you may either specify a day of the month such as 1, 2, ... 31, or you may specify a day of the week such as Monday, Tuesday, ... Sunday. Finally, you may also specify a week qualifier to restrict the schedule to the first, second, third, fourth, or fifth week of the month.

For example, if you specify only a day of the week, such as **Tuesday** the Job will be run every hour of every Tuesday of every Month. That is the **month** and **hour** remain set to the defaults of every month and all hours.

Note, by default with no other specification, your job will run at the beginning of every hour. If you wish your job to run more than once in any given hour, you will need to specify multiple **run** specifications each with a different minute.

The date/time to run the Job can be specified in the following way in pseudo-BNF:

```
<void-keyword>    = on
<at-keyword>      = at
<week-keyword>    = 1st | 2nd | 3rd | 4th | 5th | first |
                    second | third | fourth | fifth
<wday-keyword>    = sun | mon | tue | wed | thu | fri | sat |
                    sunday | monday | tuesday | wednesday |
                    thursday | friday | saturday
<week-of-year-keyword> = w00 | w01 | ... w52 | w53
<month-keyword>   = jan | feb | mar | apr | may | jun | jul |
                    aug | sep | oct | nov | dec | january |
                    february | ... | december
<daily-keyword>   = daily
<weekly-keyword>  = weekly
<monthly-keyword> = monthly
<hourly-keyword>  = hourly
<digit>           = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<number>          = <digit> | <digit><number>
<12hour>          = 0 | 1 | 2 | ... 12
<hour>            = 0 | 1 | 2 | ... 23
<minute>          = 0 | 1 | 2 | ... 59
<day>             = 1 | 2 | ... 31
<time>            = <hour>:<minute> |
                    <12hour>:<minute>am |
                    <12hour>:<minute>pm
<time-spec>       = <at-keyword> <time> |
                    <hourly-keyword>
<date-keyword>    = <void-keyword>  <weekly-keyword>
<day-range>       = <day>-<day>
<month-range>     = <month-keyword>-<month-keyword>
<wday-range>      = <wday-keyword>-<wday-keyword>
<range>           = <day-range> | <month-range> |
                          <wday-range>
<date>            = <date-keyword> | <day> | <range>
<date-spec>       = <date> | <date-spec>
<day-spec>        = <day> | <wday-keyword> |
                    <day> | <wday-range> |
                    <week-keyword> <wday-keyword> |
                    <week-keyword> <wday-range> |
                    <daily-keyword>
<month-spec>      = <month-keyword> | <month-range> |
                    <monthly-keyword>
<date-time-spec>  = <month-spec> <day-spec> <time-spec>
```

Note, the Week of Year specification wnn follows the ISO standard definition of the week of the year, where Week 1 is the week in which the first Thursday of the year occurs, or alternatively, the week which contains the 4th of January. Weeks are numbered w01 to w53. w00 for Bacula is the week that precedes the first ISO week (i.e. has the first few days of the year if any occur before Thursday). w00 is not defined by the ISO specification. A week starts with Monday and ends with Sunday.

According to the NIST (US National Institute of Standards and Technology), 12am and 12pm are ambiguous and can be defined to anything. However, 12:01am is the same as 00:01 and 12:01pm is the same as 12:01, so Bacula defines 12am as 00:00 (midnight) and 12pm as 12:00 (noon). You can avoid this abiguity (confusion) by using 24 hour time specifications (i.e. no am/pm). This is the definition in Bacula version 2.0.3 and later.

An example schedule resource that is named **WeeklyCycle** and runs a job with level full each Sunday at 2:05am and an incremental job Monday through Saturday at 2:05am is:

```
Schedule {
  Name = "WeeklyCycle"
  Run = Level=Full sun at 2:05
  Run = Level=Incremental mon-sat at 2:05
}
```

An example of a possible monthly cycle is as follows:

```
Schedule {
  Name = "MonthlyCycle"
  Run = Level=Full Pool=Monthly 1st sun at 2:05
  Run = Level=Differential 2nd-5th sun at 2:05
  Run = Level=Incremental Pool=Daily mon-sat at 2:05
}
```

The first of every month:

```
Schedule {
  Name = "First"
  Run = Level=Full on 1 at 2:05
  Run = Level=Incremental on 2-31 at 2:05
}
```

Every 10 minutes:

```
Schedule {
  Name = "TenMinutes"
  Run = Level=Full hourly at 0:05
  Run = Level=Full hourly at 0:15
  Run = Level=Full hourly at 0:25
  Run = Level=Full hourly at 0:35
  Run = Level=Full hourly at 0:45
  Run = Level=Full hourly at 0:55
}
```

5.6 Technical Notes on Schedules

Internally Bacula keeps a schedule as a bit mask. There are six masks and a minute field to each schedule. The masks are hour, day of the month (mday), month, day of the week (wday), week of the month (wom), and week of the year (woy). The schedule is initialized to have the bits of each of these masks set, which means that at the beginning of every hour, the job will run. When you specify a month for the first time, the mask will be cleared and the bit corresponding to your selected month will be selected. If you specify a second month, the bit corresponding to it will also be added to the mask. Thus when Bacula checks the masks to see if the bits are set corresponding to the current time, your job will run only in the two months you have set. Likewise, if you set a time (hour), the hour mask will be cleared, and the hour you specify will be set in the bit mask and the minutes will be stored in the minute field.

For any schedule you have defined, you can see how these bits are set by doing a **show schedules** command in the Console program. Please note that the bit mask is zero based, and Sunday is the first day of the week (bit zero).

-

5.7 The FileSet Resource

The FileSet resource defines what files are to be included or excluded in a backup job. A **FileSet** resource is required for each backup Job. It consists of a list of files or directories to be included, a list of files or directories to be excluded and the various backup options such as compression, encryption, and signatures that are to be applied to each file.

Any change to the list of the included files will cause Bacula to automatically create a new FileSet (defined by the name and an MD5 checksum of the Include/Exclude contents). Each time a new FileSet is created, Bacula will ensure that the next backup is always a Full save.

Bacula is designed to handle most character sets of the world, US ASCII, German, French, Chinese, ... However, it does this by encoding everything in UTF-8, and it expects all configuration files (including those read on Win32 machines) to be in UTF-8 format. UTF-8 is typically the default on Linux machines, but not on all Unix machines, nor on Windows, so you must take some care to ensure that your locale is set properly before starting Bacula. On most modern Win32 machines, you can edit the conf files with **notebook** and choose output encoding UTF-8.

To ensure that Bacula configuration files can be correctly read including foreign characters the bf LANG environment variable must end in **.UTF-8**. An full example is **en_US.UTF-8**. The exact syntax may vary a bit from OS to OS, and exactly how you define it will also vary.

Bacula assumes that all filenames are in UTF-8 format on Linux and Unix machines. On Win32 they are in Unicode (UTF-16), and will be automatically converted to UTF-8 format.

FileSet Start of the FileSet resource. One **FileSet** resource must be defined for each Backup job.

Name = <name> The name of the FileSet resource. This directive is required.

Ignore FileSet Changes = <yes|no> Normally, if you modify the FileSet Include or Exclude lists, the next backup will be forced to a Full so that Bacula can guarantee that any additions or deletions are properly saved.

We strongly recommend against setting this directive to yes, since doing so may cause you to have an incomplete set of backups.

If this directive is set to **yes**, any changes you make to the FileSet Include or Exclude lists, will not force a Full during subsequent backups.

The default is **no**, in which case, if you change the Include or Exclude, Bacula will force a Full backup to ensure that everything is properly backed up.

Enable VSS = <yes|no> If this directive is set to **yes** the File daemon will be notified that the user wants to use a Volume Shadow Copy Service (VSS) backup for this job. The default is **yes**. This directive is effective only for VSS enabled Win32 File daemons. It permits a consistent copy of open files to be made for cooperating writer applications, and for applications that are not VSS away, Bacula can at least copy open files. For more information, please see the Windows chapter of this manual.

Include { Options {<file-options>} ...; <file-list> }

Options { <file-options> }

Exclude { <file-list> }

The Include resource must contain a list of directories and/or files to be processed in the backup job. Normally, all files found in all subdirectories of any directory in the Include File list will be backed up. Note, see below for the definition of <file-list>. The Include resource may also contain one or more Options resources that specify options such as compression to be applied to all or any subset of the files found when processing the file-list for backup. Please see below for more details concerning Options resources.

There can be any number of **Include** resources within the FileSet, each having its own list of directories or files to be backed up and the backup options defined by one or more Options resources. The **file-list** consists of one file or directory name per line. Directory names should be specified without a trailing slash with Unix path notation.

Windows users, please take note to specify directories (even c:/...) in Unix path notation. If you use Windows conventions, you will most likely not be able to restore your files due to the fact that the Windows path separator was defined as an escape character long before Windows existed, and Bacula adheres to that convention (i.e.
means the next character appears as itself).

You should always specify a full path for every directory and file that you list in the FileSet. In addition, on Windows machines, you should **always** prefix the directory or filename with the drive specification (e.g. **c:/xxx**) using Unix directory name separators (forward slash). The drive letter itself can be upper or lower case (e.g. c:/xxx or C:/xxx).

Bacula's default for processing directories is to recursively descend in the directory saving all files and subdirectories. Bacula will not by default cross filesystems (or mount points in Unix parlance). This means that if you specify the root partition (e.g. /), Bacula will save only the root partition and not any of the other mounted filesystems. Similarly on Windows systems, you must explicitly specify each of the drives you want saved (e.g. **c:/** and **d:/** ...). In addition, at least for Windows systems, you will most likely want to enclose each specification within double quotes particularly if the directory (or file) name contains spaces. The **df** command on Unix systems will show you which mount points you must specify to save everything. See below for an example.

Take special care not to include a directory twice or Bacula will backup the same files two times wasting a lot of space on your archive device. Including a directory twice is very easy to do. For example:

```
Include {
  File = /
  File = /usr
  Options { compression=GZIP }
}
```

on a Unix system where /usr is a subdirectory (rather than a mounted filesystem) will cause /usr to be backed up twice. In this case, on Bacula versions prior to 1.32f-5-09Mar04 due to a bug, you will not be able to restore hard linked files that were backed up twice.

If you have used Bacula prior to version 1.36.3, you will note three things in the new FileSet syntax:

1. There is no equal sign (=) after the Include and before the opening brace ({). The same is true for the Exclude.
2. Each directory (or filename) to be included or excluded is preceded by a **File =**. Previously they were simply listed on separate lines.
3. The options that previously appeared on the Include line now must be specified within their own Options resource.
4. The Exclude resource does not accept Options.
5. When using wild-cards or regular expressions, directory names are always terminated with a slash (/) and filenames have no trailing slash.

The Options resource is optional, but when specified, it will contain a list of **keyword=value** options to be applied to the file-list. See below for the definition of file-list. Multiple Options resources may be specified one after another. As the files are found in the specified directories, the Options will applied to the filenames to determine if and how the file should be backed up. The wildcard and regular expression pattern matching parts of the Options resources are checked in the order they are specified in the FileSet until the first one that matches. Once one matches, the compression and other flags within the Options specification will apply to the pattern matched.

A key point is that in the absence of an Option or no other Option is matched, every file is accepted for backing up. This means that if you want to exclude something, you must explicitly specify an Option with an **exclude = yes** and some pattern matching.

Once Bacula determines that the Options resource matches the file under consideration, that file will be saved without looking at any other Options resources that may be present. This means that any wild cards must appear before an Options resource without wild cards.

If for some reason, Bacula checks all the Options resources to a file under consideration for backup, but there are no matches (generally because of wild cards that don't match), Bacula as a default will then backup the file. This is quite logical if you consider the case of no Options clause is specified, where you want everything to be backed up, and it is important to keep in mind when excluding as mentioned above.

However, one additional point is that in the case that no match was found, Bacula will use the options found in the last Options resource. As a consequence, if you want a particular set of "default" options, you should put them in an Options resource after any other Options.

It is a good idea to put all your wild-card and regex expressions inside double quotes to prevent conf file scanning problems.

This is perhaps a bit overwhelming, so there are a number of examples included below to illustrate how this works.

You find yourself using a lot of Regex statements, which will cost quite a lot of CPU time, we recommend you simplify them if you can, or better yet convert them to Wild statements which are much more efficient.

The directives within an Options resource may be one of the following:

compression=GZIP All files saved will be software compressed using the GNU ZIP compression format. The compression is done on a file by file basis by the File daemon. If there is a problem reading the tape in a single record of a file, it will at most affect that file and none of the other files on the tape. Normally this option is **not** needed if you have a modern tape drive as the drive will do its own compression. In fact, if you specify software compression at the same time you have hardware compression turned on, your files may actually take more space on the volume.

Software compression is very important if you are writing your Volumes to a file, and it can also be helpful if you have a fast computer but a slow network, otherwise it is generally better to rely your tape drive's hardware compression. As noted above, it is not generally a good idea to do both software and hardware compression.

Specifying **GZIP** uses the default compression level 6 (i.e. **GZIP** is identical to **GZIP6**). If you want a different compression level (1 through 9), you can specify it by appending the level number with no intervening spaces to **GZIP**. Thus **compression=GZIP1** would give minimum compression but the fastest algorithm, and **compression=GZIP9** would give the highest level of compression, but requires more computation. According to the GZIP documentation, compression levels greater than six generally give very little extra compression and are rather CPU intensive.

signature=SHA1 An SHA1 signature will be computed for all The SHA1 algorithm is purported to be some what slower than the MD5 algorithm, but at the same time is significantly better from a cryptographic point of view (i.e. much fewer collisions, much lower probability of being hacked.) It adds four more bytes than the MD5 signature. We strongly recommend that either this option or MD5 be specified as a default for all files. Note, only one of the two options MD5 or SHA1 can be computed for any file.

signature=MD5 An MD5 signature will be computed for all files saved. Adding this option generates about 5% extra overhead for each file saved. In addition to the additional CPU time, the MD5 signature adds 16 more bytes per file to your catalog. We strongly recommend that this option or the SHA1 option be specified as a default for all files.

basejob=<options> The options letters specified are used when running a **Backup Level=Full** with BaseJobs. The options letters are the same than in the **verify=** option below.

accurate=<options> The options letters specified are used when running a **Backup Level=Incremental/Differential** in Accurate mode. The options letters are the same than in the **verify=** option below.

verify=<options> The options letters specified are used when running a **Verify Level=Catalog** as well as the **DiskToCatalog** level job. The options letters may be any combination of the following:

i compare the inodes

p compare the permission bits

n compare the number of links

u compare the user id

g compare the group id

s compare the size

a compare the access time

m compare the modification time (st_mtime)

c compare the change time (st_ctime)

d report file size decreases

5 compare the MD5 signature

1 compare the SHA1 signature

A useful set of general options on the **Level=Catalog** or **Level=DiskToCatalog** verify is **pins5** i.e. compare permission bits, inodes, number of links, size, and MD5 changes.

onefs=yes|no If set to **yes** (the default), **Bacula** will remain on a single file system. That is it will not backup file systems that are mounted on a subdirectory. If you are using a *nix system, you may not even be aware that there are several different filesystems as they are often automatically mounted by the OS (e.g. /dev, /net, /sys, /proc, ...). With Bacula 1.38.0 or later, it will inform you when it decides not to traverse into another filesystem. This can be very useful if you forgot to backup a particular partition. An example of the informational message in the job report is:

```
rufus-fd: /misc is a different filesystem. Will not descend from / into /misc
rufus-fd: /net is a different filesystem. Will not descend from / into /net
rufus-fd: /var/lib/nfs/rpc_pipefs is a different filesystem. Will not descend from /var/lib/nfs into /var/lib/nfs/rpc_pipe
rufus-fd: /selinux is a different filesystem. Will not descend from / into /selinux
rufus-fd: /sys is a different filesystem. Will not descend from / into /sys
rufus-fd: /dev is a different filesystem. Will not descend from / into /dev
rufus-fd: /home is a different filesystem. Will not descend from / into /home
```

Note: in previous versions of Bacula, the above message was of the form:

```
Filesystem change prohibited. Will not descend into /misc
```

If you wish to backup multiple filesystems, you can explicitly list each filesystem you want saved. Otherwise, if you set the onefs option to **no**, Bacula will backup all mounted file systems (i.e. traverse mount points) that are found within the **FileSet**. Thus if you have NFS or Samba file systems mounted on a directory listed in your FileSet, they will also be backed up. Normally, it is preferable to set **onefs=yes** and to explicitly name each filesystem you want backed up. Explicitly naming the filesystems you want backed up avoids the possibility of getting into a infinite loop recursing filesystems. Another possibility is to use **onefs=no** and to set **fstype=ext2, ...**. See the example below for more details.

If you think that Bacula should be backing up a particular directory and it is not, and you have **onefs=no** set, before you complain, please do:

```
  stat /
  stat <filesystem>
```

where you replace **filesystem** with the one in question. If the **Device:** number is different for / and for your filesystem, then they are on different filesystems. E.g.

```
stat /
  File: '/'
  Size: 4096            Blocks: 16         IO Block: 4096   directory
Device: 302h/770d       Inode: 2           Links: 26
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2005-11-10 12:28:01.000000000 +0100
Modify: 2005-09-27 17:52:32.000000000 +0200
Change: 2005-09-27 17:52:32.000000000 +0200

stat /net
  File: '/home'
  Size: 4096            Blocks: 16         IO Block: 4096   directory
Device: 308h/776d       Inode: 2           Links: 7
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2005-11-10 12:28:02.000000000 +0100
Modify: 2005-11-06 12:36:48.000000000 +0100
Change: 2005-11-06 12:36:48.000000000 +0100
```

Also be aware that even if you include **/home** in your list of files to backup, as you most likely should, you will get the informational message that "/home is a different filesystem" when Bacula is processing the / directory. This message does not indicate an error. This message means that while examining the **File =** referred to in the second part of the message, Bacula will not descend into the directory mentioned in the first part of the message. However, it is possible that the separate filesystem will be backed up despite the message. For example, consider the following FileSet:

```
File = /
File = /var
```

where **/var** is a separate filesystem. In this example, you will get a message saying that Bacula will not decend from / into **/var**. But it is important to realise that Bacula will descend into **/var** from the second File directive shown above. In effect, the warning is bogus, but it is supplied to alert you to possible omissions from your FileSet. In this example, **/var** will be backed up. If you changed the FileSet such that it did not specify **/var**, then **/var** will not be backed up.

honor nodump flag=<yes|no> If your file system supports the **nodump** flag (e. g. most BSD-derived systems) Bacula will honor the setting of the flag when this option is set to **yes**. Files having this flag set will not be included in the backup and will not show up in the catalog. For directories with the **nodump** flag set recursion is turned off and the directory will be listed in the catalog. If the **honor nodump flag** option is not defined or set to **no** every file and directory will be eligible for backup.

portable=yes|no If set to **yes** (default is **no**), the Bacula File daemon will backup Win32 files in a portable format, but not all Win32 file attributes will be saved and restored. By default, this option is set to **no**, which means that on Win32 systems, the data will be backed up using Windows API calls and on WinNT/2K/XP, all the security and ownership attributes will be properly backed up (and restored). However this format is not portable to other systems – e.g. Unix, Win95/98/Me. When backing up Unix systems, this option is ignored, and unless you have a specific need to have portable backups, we recommend accept the default (**no**) so that the maximum information concerning your files is saved.

recurse=yes|no If set to **yes** (the default), Bacula will recurse (or descend) into all subdirectories found unless the directory is explicitly excluded using an **exclude** definition. If you set **recurse=no**, Bacula will save the subdirectory entries, but not descend into the subdirectories, and thus will not save the files or directories contained in the subdirectories. Normally, you will want the default (**yes**).

sparse=yes|no Enable special code that checks for sparse files such as created by ndbm. The default is **no**, so no checks are made for sparse files. You may specify **sparse=yes** even on files that are not sparse file. No harm will be done, but there will be a small additional overhead to check for buffers of all zero, and a small additional amount of space on the output archive will be used to save the seek address of each non-zero record read.

Restrictions: Bacula reads files in 32K buffers. If the whole buffer is zero, it will be treated as a sparse block and not written to tape. However, if any part of the buffer is non-zero, the whole buffer will be written to tape, possibly including some disk sectors (generally 4098 bytes) that are all zero. As a consequence, Bacula's detection of sparse blocks is in 32K increments rather than the system block size. If anyone considers this to be a real problem, please send in a request for change with the reason.

If you are not familiar with sparse files, an example is say a file where you wrote 512 bytes at address zero, then 512 bytes at address 1 million. The operating system will allocate only two blocks, and the empty space or hole will have nothing allocated. However, when you read the sparse file and read the addresses where nothing was written, the OS will return all zeros as if the space were allocated, and if you backup such a file, a lot of space will be used to write zeros to the volume. Worse yet, when you restore the file, all the previously empty space will now be allocated using much more disk space. By turning on the **sparse** option, Bacula will specifically look for empty space in the file, and any empty space will not be written to the Volume, nor will it be restored. The price to pay for this is that Bacula must search each block it reads before writing it. On a slow system, this may be important. If you suspect you have sparse files, you should benchmark the difference or set sparse for only those files that are really sparse.

readfifo=yes|no If enabled, tells the Client to read the data on a backup and write the data on a restore to any FIFO (pipe) that is explicitly mentioned in the FileSet. In this case, you must have a program already running that writes into the FIFO for a backup or reads from the FIFO on a restore. This can be accomplished with the **RunBeforeJob** directive. If this is not the case, Bacula will hang indefinitely on reading/writing the FIFO. When this is not enabled (default), the Client simply saves the directory entry for the FIFO.

Unfortunately, when Bacula runs a RunBeforeJob, it waits until that script terminates, and if the script accesses the FIFO to write into the it, the Bacula job will block and everything will stall. However, Vladimir Stavrinov as supplied tip that allows this feature to work correctly. He simply adds the following to the beginning of the RunBeforeJob script:

```
exec > /dev/null
```

noatime=yes|no If enabled, and if your Operating System supports the O_NOATIME file open flag, Bacula will open all files to be backed up with this option. It makes it possible to read a file without updating the inode atime (and also without the inode ctime update which happens if you try to set the atime back to its previous value). It also prevents a race condition when two programs are reading the same file, but only one does not want to change the atime. It's most useful for backup programs and file integrity checkers (and bacula can fit on both categories).

This option is particularly useful for sites where users are sensitive to their MailBox file access time. It replaces both the **keepatime** option without the inconveniences of that option (see below).

If your Operating System does not support this option, it will be silently ignored by Bacula.

mtimeonly=yes|no If enabled, tells the Client that the selection of files during Incremental and Differential backups should based only on the st_mtime value in the stat() packet. The default is **no** which means that the selection of files to be backed up will be based on both the st_mtime and the st_ctime values. In general, it is not recommended to use this option.

keepatime=yes|no The default is **no**. When enabled, Bacula will reset the st_atime (access time) field of files that it backs up to their value prior to the backup. This option is not generally recommended as there are very few programs that use st_atime, and the backup overhead is increased because of the additional system call necessary to reset the times. However, for some files, such as mailboxes, when Bacula backs up the file, the user will notice that someone (Bacula) has accessed the file. In this, case keepatime can be useful. (I'm not sure this works on Win32).

Note, if you use this feature, when Bacula resets the access time, the change time (st_ctime) will automatically be modified by the system, so on the next incremental job, the file will be backed up even if it has not changed. As a consequence, you will probably also want to use **mtimeonly = yes** as well as keepatime (thanks to Rudolf Cejka for this tip).

checkfilechanges=yes|no On versions 2.0.4 or greater, if enabled, the Client will checks size, age of each file after their backup to see if they have changed during backup. If time or size mismatch, an error will raise.

```
zog-fd: Client1.2007-03-31_09.46.21 Error: /tmp/test mtime changed during backup.
```

In general, it is recommended to use this option.

hardlinks=yes|no When enabled (default), this directive will cause hard links to be backed up. However, the File daemon keeps track of hard linked files and will backup the data only once. The process of keeping track of the hard links can be quite expensive if you have lots of them (tens of thousands or more). This doesn't occur on normal Unix systems, but if you use a program like BackupPC, it can create hundreds of thousands, or even millions of hard links. Backups become very long and the File daemon will consume a lot of CPU power checking hard links. In such a case, set **hardlinks=no** and hard links will not be backed up. Note, using this option will most likely backup more data and on a restore the file system will not be restored identically to the original.

wild=<string> Specifies a wild-card string to be applied to the filenames and directory names. Note, if **Exclude** is not enabled, the wild-card will select which files are to be included. If **Exclude=yes** is specified, the wild-card will select which files are to be excluded. Multiple wild-card directives may be specified, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the Utilities chapter of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this manual. It is recommended to enclose the string in double quotes.

wilddir=<string> Specifies a wild-card string to be applied to directory names only. No filenames will be matched by this directive. Note, if **Exclude** is not enabled, the wild-card will select directories to be included. If **Exclude=yes** is specified, the wild-card will select which directories are to be excluded. Multiple wild-card directives may be specified, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

It is recommended to enclose the string in double quotes.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the Utilities chapter of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this manual. An example of excluding with the WildDir option on Win32 machines is presented below.

wildfile=<string> Specifies a wild-card string to be applied to non-directories. That is no directory entries will be matched by this directive. However, note that the match is done against the full path and filename, so your wild-card string must take into account that filenames are preceded by the full path. If **Exclude** is not enabled, the wild-card will select which files are to be included. If **Exclude=yes** is specified, the wild-card will select which files are to be excluded. Multiple wild-card directives may be specified, and they will be applied in turn until the first one that matches.

It is recommended to enclose the string in double quotes.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the Utilities chapter of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this manual. An example of excluding with the WildFile option on Win32 machines is presented below.

regex=<string> Specifies a POSIX extended regular expression to be applied to the filenames and directory names, which include the full path. If **Exclude** is not enabled, the regex will select which files are to be included. If **Exclude=yes** is specified, the regex will select which files are to be excluded. Multiple regex directives may be specified within an Options resource, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program. Please see the Utilities chapter of this manual for more. You can also test your full FileSet definition by using the estimate command in the Console chapter of this manual.

You find yourself using a lot of Regex statements, which will cost quite a lot of CPU time, we recommend you simplify them if you can, or better yet convert them to Wild statements which are much more efficient.

regexfile=<string> Specifies a POSIX extended regular expression to be applied to non-directories. No directories will be matched by this directive. However, note that the match is done against the full path and filename, so your regex string must take into account that filenames are preceded by the full path. If **Exclude** is not enabled, the regex will select which files are to be included. If **Exclude=yes** is specified, the regex will select which files are to be excluded. Multiple regex directives may be specified, and they will be applied in turn until the first one that matches.

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program. Please see the Utilities chapter of this manual for more.

regexdir=<string> Specifies a POSIX extended regular expression to be applied to directory names only. No filenames will be matched by this directive. Note, if **Exclude** is not enabled, the regex will select directories files are to be included. If **Exclude=yes** is specified, the regex will select which files are to be excluded. Multiple regex directives may be specified, and they will be applied in turn until the first one that matches. Note, if you exclude a directory, no files or directories below it will be matched.

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program. Please see the Utilities chapter of this manual for more.

exclude=yes|no The default is **no**. When enabled, any files matched within the Options will be excluded from the backup.

aclsupport=yes|no The default is **no**. If this option is set to yes, and you have the POSIX **libacl** installed on your system, Bacula will backup the file and directory UNIX Access Control Lists (ACL) as defined in IEEE Std 1003.1e draft 17 and "POSIX.1e" (abandoned). This feature is available on UNIX only and depends on the ACL library. Bacula is automatically compiled with ACL support if the **libacl** library is installed on your system (shown in config.out). While restoring the files Bacula will try to restore the ACLs, if there is no ACL support available on the system, Bacula restores the files and directories but not the ACL information. Please note, if you backup an EXT3 or XFS filesystem with ACLs, then you restore them to a different filesystem (perhaps reiserfs) that does not have ACLs, the ACLs will be ignored.

ignore case=yes|no The default is **no**. On Windows systems, you will almost surely want to set this to **yes**. When this directive is set to **yes** all the case of character will be ignored in wild-card and regex comparisons. That is an uppercase A will match a lowercase a.

fstype=filesystem-type This option allows you to select files and directories by the filesystem type. The permitted filesystem-type names are:

ext2, jfs, ntfs, proc, reiserfs, xfs, usbdevfs, sysfs, smbfs, iso9660. For ext3 systems, use ext2.

You may have multiple Fstype directives, and thus permit matching of multiple filesystem types within a single Options resource. If the type specified on the fstype directive does not match the filesystem for a particular directive, that directory will not be backed up. This directive can be used to prevent backing up non-local filesystems. Normally, when you use this directive, you would also set **onefs=no** so that Bacula will traverse filesystems.

This option is not implemented in Win32 systems.

DriveType=Windows-drive-type This option is effective only on Windows machines and is somewhat similar to the Unix/Linux **fstype** described above, except that it allows you to select what Windows drive types you want to allow. By default all drive types are accepted.

The permitted drivetype names are:

removable, fixed, remote, cdrom, ramdisk

You may have multiple Driveype directives, and thus permit matching of multiple drive types within a single Options resource. If the type specified on the drivetype directive does not match the filesystem for a particular directive, that directory will not be backed up. This directive can be used to prevent backing up non-local filesystems. Normally, when you use this directive, you would also set **onefs=no** so that Bacula will traverse filesystems.

This option is not implemented in Unix/Linux systems.

hfsplussupport=yes|no This option allows you to turn on support for Mac OSX HFS plus finder information.

strippath=<integer> This option will cause **integer** paths to be stripped from the front of the full path/filename being backed up. This can be useful if you are migrating data from another vendor or if you have taken a snapshot into some subdirectory. This directive can cause your filenames to be overlayed with regular backup data, so should be used only by experts and with great care.

<file-list> is a list of directory and/or filename names specified with a **File =** directive. To include names containing spaces, enclose the name between double-quotes. Wild-cards are not interpreted in file-lists. They can only be specified in Options resources.

There are a number of special cases when specifying directories and files in a **file-list**. They are:

- Any name preceded by an at-sign (@) is assumed to be the name of a file, which contains a list of files each preceded by a "File =". The named file is read once when the configuration file is parsed during the Director startup. Note, that the file is read on the Director's machine and not on the Client's. In fact, the @filename can appear anywhere within the conf file where a token would be read, and the contents of the named file will be logically inserted in the place of the @filename. What must be in the file depends on the location the @filename is specified in the conf file. For example:

```
Include {
  Options { compression=GZIP }
  @/home/files/my-files
}
```

- Any name beginning with a vertical bar (|) is assumed to be the name of a program. This program will be executed on the Director's machine at the time the Job starts (not when the Director reads the configuration file), and any output from that program will be assumed to be a list of files or directories, one per line, to be included. Before submitting the specified command bacula will performe character substitution.

  This allows you to have a job that, for example, includes all the local partitions even if you change the partitioning by adding a disk. The examples below show you how to do this. However, please note two things:
  1. if you want the local filesystems, you probably should be using the new **fstype** directive, which was added in version 1.36.3 and set **onefs=no**.

  2. the exact syntax of the command needed in the examples below is very system dependent. For example, on recent Linux systems, you may need to add the -P option, on FreeBSD systems, the options will be different as well.

  In general, you will need to prefix your command or commands with a **sh -c** so that they are invoked by a shell. This will not be the case if you are invoking a script as in the second example below. Also, you must take care to escape (precede with a \) wild-cards, shell character, and to ensure that any spaces in your command are escaped as well. If you use a single quotes (') within a double quote ("), Bacula will treat everything between the single quotes as one field so it will not be necessary to escape the spaces. In general, getting all the quotes and escapes correct is a real pain as you can see by the next example. As a consequence, it is often easier to put everything in a file and simply use the file name within Bacula. In that case the **sh -c** will not be necessary providing the first line of the file is **#!/bin/sh**.

  As an example:

```
Include {
   Options { signature = SHA1 }
   File = "|sh -c 'df -l | grep \"^/dev/hd[ab]\" | grep -v \".*/tmp\" \
      | awk \"{print \\$6}\"'"
}
```

  will produce a list of all the local partitions on a Red Hat Linux system. Note, the above line was split, but should normally be written on one line. Quoting is a real problem because you must quote for Bacula which consists of preceding every \ and every " with a \, and you must also quote for the shell command. In the end, it is probably easier just to execute a small file with:

```
Include {
  Options {
    signature=MD5
  }
  File = "|my_partitions"
}
```

  where my_partitions has:

```
#!/bin/sh
df -l | grep "^/dev/hd[ab]" | grep -v ".*/tmp" \
      | awk "{print \$6}"
```

  If the vertical bar (—) in front of my_partitions is preceded by a backslash as in \—, the program will be executed on the Client's machine instead of on the Director's machine. Please note that if the filename is given within quotes, you will need to use two slashes. An example, provided by John Donagher, that backs up all the local UFS partitions on a remote system is:

```
FileSet {
  Name = "All local partitions"
  Include {
    Options { signature=SHA1; onefs=yes; }
    File = "\\|bash -c \"df -klF ufs | tail +2 | awk '{print \$6}'\""
  }
}
```

The above requires two backslash characters after the double quote (one preserves the next one). If you are a Linux user, just change the **ufs** to **ext3** (or your preferred filesystem type), and you will be in business.

If you know what filesystems you have mounted on your system, e.g. for Red Hat Linux normally only ext2 and ext3, you can backup all local filesystems using something like:

```
Include {
   Options { signature = SHA1; onfs=no; fstype=ext2 }
   File = /
}
```

- Any file-list item preceded by a less-than sign (<) will be taken to be a file. This file will be read on the Director's machine (see below for doing it on the Client machine) at the time the Job starts, and the data will be assumed to be a list of directories or files, one per line, to be included. The names should start in column 1 and should not be quoted even if they contain spaces. This feature allows you to modify the external file and change what will be saved without stopping and restarting Bacula as would be necessary if using the @ modifier noted above. For example:

```
Include {
  Options { signature = SHA1 }
  File = "</home/files/local-filelist"
}
```

  If you precede the less-than sign (<) with a backslash as in \<, the file-list will be read on the Client machine instead of on the Director's machine. Please note that if the filename is given within quotes, you will need to use two slashes.

```
Include {
  Options { signature = SHA1 }
  File = "\\</home/xxx/filelist-on-client"
}
```

- If you explicitly specify a block device such as **/dev/hda1**, then Bacula (starting with version 1.28) will assume that this is a raw partition to be backed up. In this case, you are strongly urged to specify a **sparse=yes** include option, otherwise, you will save the whole partition rather than just the actual data that the partition contains. For example:

```
Include {
  Options { signature=MD5; sparse=yes }
  File = /dev/hd6
}
```

  will backup the data in device /dev/hd6. Note, the bf /dev/hd6 must be the raw partition itself. Bacula will not back it up as a raw device if you specify a symbolic link to a raw device such as my be created by the LVM Snapshot utilities.

  Ludovic Strappazon has pointed out that this feature can be used to backup a full Microsoft Windows disk. Simply boot into the system using a Linux Rescue disk, then load a statically linked Bacula as described in the Disaster Recovery Using Bacula chapter of this manual. Then save the whole disk partition. In the case of a disaster, you can then restore the desired partition by again booting with the rescue disk and doing a restore of the partition.

- If you explicitly specify a FIFO device name (created with mkfifo), and you add the option **readfifo=yes** as an option, Bacula will read the FIFO and back its data up to the Volume. For example:

```
Include {
  Options {
    signature=SHA1
    readfifo=yes
  }
  File = /home/abc/fifo
}
```

if **/home/abc/fifo** is a fifo device, Bacula will open the fifo, read it, and store all data thus obtained on the Volume. Please note, you must have a process on the system that is writing into the fifo, or Bacula will hang, and after one minute of waiting, Bacula will give up and go on to the next file. The data read can be anything since Bacula treats it as a stream.

This feature can be an excellent way to do a "hot" backup of a very large database. You can use the **RunBeforeJob** to create the fifo and to start a program that dynamically reads your database and writes it to the fifo. Bacula will then write it to the Volume. Be sure to read the readfifo section that gives a tip to ensure that the RunBeforeJob does not block Bacula.

During the restore operation, the inverse is true, after Bacula creates the fifo if there was any data stored with it (no need to explicitly list it or add any options), that data will be written back to the fifo. As a consequence, if any such FIFOs exist in the fileset to be restored, you must ensure that there is a reader program or Bacula will block, and after one minute, Bacula will time out the write to the fifo and move on to the next file.

- A file-list may not contain wild-cards. Use directives in the Options resource if you wish to specify wild-cards or regular expression matching.
- The **ExcludeDirContaining = <filename>** is a directive that can be added to the Include section of the FileSet resource. If the specified filename (**filename-string**) is found on the Client in any directory to be backed up, the whole directory will be ignored (not backed up). For example:

```
# List of files to be backed up
FileSet {
  Name = "MyFileSet"
  Include {
    Options {
      signature = MD5
    }
    File = /home
    Exclude Dir Containing = .excludeme
  }
}
```

But in /home, there may be hundreds of directories of users and some people want to indicate that they don't want to have certain directories backed up. For example, with the above FileSet, if the user or sysadmin creates a file named **.excludeme** in specific directories, such as

```
/home/user/www/cache/.excludeme
/home/user/temp/.excludeme
```

then Bacula will not backup the two directories named:

```
/home/user/www/cache
/home/user/temp
```

NOTE: subdirectories will not be backed up. That is, the directive applies to the two directories in question and any children (be they files, directories, etc).

5.8 FileSet Examples

The following is an example of a valid FileSet resource definition. Note, the first Include pulls in the contents of the file **/etc/backup.list** when Bacula is started (i.e. the @), and that file must have each filename to be backed up preceded by a **File =** and on a separate line.

```
FileSet {
  Name = "Full Set"
  Include {
```

```
    Options {
      Compression=GZIP
      signature=SHA1
      Sparse = yes
    }
    @/etc/backup.list
  }
  Include {
     Options {
        wildfile = "*.o"
        wildfile = "*.exe"
        Exclude = yes
     }
     File = /root/myfile
     File = /usr/lib/another_file
  }
}
```

In the above example, all the files contained in /etc/backup.list will be compressed with GZIP compression, an SHA1 signature will be computed on the file's contents (its data), and sparse file handling will apply.

The two directories /root/myfile and /usr/lib/another_file will also be saved without any options, but all files in those directories with the extensions **.o** and **.exe** will be excluded.

Let's say that you now want to exclude the directory /tmp. The simplest way to do so is to add an exclude directive that lists /tmp. The example above would then become:

```
FileSet {
  Name = "Full Set"
  Include {
    Options {
      Compression=GZIP
      signature=SHA1
      Sparse = yes
    }
    @/etc/backup.list
  }
  Include {
     Options {
        wildfile = "*.o"
        wildfile = "*.exe"
        Exclude = yes
     }
     File = /root/myfile
     File = /usr/lib/another_file
  }
  Exclude {
     File = /tmp
  }
}
```

You can add wild-cards to the File directives listed in the Exclude directory, but you need to take care because if you exclude a directory, it and all files and directories below it will also be excluded.

Now lets take a slight variation on the above and suppose you want to save all your whole filesystem except **/tmp**. The problem that comes up is that Bacula will not normally cross from one filesystem to another. Doing a **df** command, you get the following output:

```
[kern@rufus k]$ df
Filesystem      1k-blocks      Used Available Use% Mounted on
/dev/hda5         5044156    439232   4348692  10% /
/dev/hda1           62193      4935     54047   9% /boot
/dev/hda9        20161172   5524660  13612372  29% /home
/dev/hda2           62217      6843     52161  12% /rescue
/dev/hda8         5044156     42548   4745376   1% /tmp
/dev/hda6         5044156   2613132   2174792  55% /usr
none               127708         0    127708   0% /dev/shm
//minimatou/c$   14099200   9895424   4203776  71% /mnt/mmatou
lmatou:/          1554264    215884   1258056  15% /mnt/matou
```

```
lmatou:/home      2478140   1589952    760072  68% /mnt/matou/home
lmatou:/usr       1981000   1199960    678628  64% /mnt/matou/usr
lpmatou:/          995116    484112    459596  52% /mnt/pmatou
lpmatou:/home    19222656   2787880  15458228  16% /mnt/pmatou/home
lpmatou:/usr      2478140   2038764    311260  87% /mnt/pmatou/usr
deuter:/          4806936     97684   4465064   3% /mnt/deuter
deuter:/home      4806904    280100   4282620   7% /mnt/deuter/home
deuter:/files    44133352  27652876  14238608  67% /mnt/deuter/files
```

And we see that there are a number of separate filesystems (/ /boot /home /rescue /tmp and /usr not to mention mounted systems). If you specify only / in your Include list, Bacula will only save the Filesystem **/dev/hda5**. To save all filesystems except **/tmp** with out including any of the Samba or NFS mounted systems, and explicitly excluding a /tmp, /proc, .journal, and .autofsck, which you will not want to be saved and restored, you can use the following:

```
FileSet {
  Name = Include_example
  Include {
    Options {
       wilddir = /proc
       wilddir = /tmp
       wildfile = "/.journal"
       wildfile = "/.autofsck"
       exclude = yes
    }
    File = /
    File = /boot
    File = /home
    File = /rescue
    File = /usr
  }
}
```

Since /tmp is on its own filesystem and it was not explicitly named in the Include list, it is not really needed in the exclude list. It is better to list it in the Exclude list for clarity, and in case the disks are changed so that it is no longer in its own partition.

Now, lets assume you only want to backup .Z and .gz files and nothing else. This is a bit trickier because Bacula by default will select everything to backup, so we must exclude everything but .Z and .gz files. If we take the first example above and make the obvious modifications to it, we might come up with a FileSet that looks like this:

```
FileSet {
  Name = "Full Set"
  Include {                    !!!!!!!!!!!!
     Options {                    This
        wildfile = "*.Z"          example
        wildfile = "*.gz"         doesn't
                                  work
     }                          !!!!!!!!!!!!
     File = /myfile
  }
}
```

The *.Z and *.gz files will indeed be backed up, but all other files that are not matched by the Options directives will automatically be backed up too (i.e. that is the default rule).

To accomplish what we want, we must explicitly exclude all other files. We do this with the following:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        wildfile = "*.Z"
        wildfile = "*.gz"
     }
```

```
    Options {
       Exclude = yes
       RegexFile = ".*"
    }
    File = /myfile
  }
}
```

The "trick" here was to add a RegexFile expression that matches all files. It does not match directory names, so all directories in /myfile will be backed up (the directory entry) and any *.Z and *.gz files contained in them. If you know that certain directories do not contain any *.Z or *.gz files and you do not want the directory entries backed up, you will need to explicitly exclude those directories. Backing up a directory entries is not very expensive.

Bacula uses the system regex library and some of them are different on different OSes. The above has been reported not to work on FreeBSD. This can be tested by using the **estimate job=job-name listing** command in the console and adapting the RegexFile expression appropriately. In a future version of Bacula, we will supply our own Regex code to avoid such system dependencies.

Please be aware that allowing Bacula to traverse or change file systems can be **very** dangerous. For example, with the following:

```
FileSet {
  Name = "Bad example"
  Include {
    Options { onefs=no }
    File = /mnt/matou
  }
}
```

you will be backing up an NFS mounted partition (**/mnt/matou**), and since **onefs** is set to **no**, Bacula will traverse file systems. Now if **/mnt/matou** has the current machine's file systems mounted, as is often the case, you will get yourself into a recursive loop and the backup will never end.

As a final example, let's say that you have only one or two subdirectories of /home that you want to backup. For example, you want to backup only subdirectories beginning with the letter a and the letter b – i.e. /home/a* and /home/b*. Now, you might first try:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        wilddir = "/home/a*"
        wilddir = "/home/b*"
     }
     File = /home
  }
}
```

The problem is that the above will include everything in /home. To get things to work correctly, you need to start with the idea of exclusion instead of inclusion. So, you could simply exclude all directories except the two you want to use:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        RegexDir = "^/home/[c-z]"
        exclude = yes
     }
     File = /home
  }
}
```

And assuming that all subdirectories start with a lowercase letter, this would work.

An alternative would be to include the two subdirectories desired and exclude everything else:

```
FileSet {
  Name = "Full Set"
  Include {
     Options {
        wilddir = "/home/a*"
        wilddir = "/home/b*"
     }
     Options {
        RegexDir = ".*"
        exclude = yes
     }
     File = /home
  }
}
```

The following example shows how to back up only the My Pictures directory inside the My Documents directory for all users in C:/Documents and Settings, i.e. everything matching the pattern:

C:/Documents and Settings/*/My Documents/My Pictures/*

To understand how this can be achieved, there are two important points to remember:

Firstly, Bacula walks over the filesystem depth-first starting from the File = lines. It stops descending when a directory is excluded, so you must include all ancestor directories of each directory containing files to be included.

Secondly, each directory and file is compared to the Options clauses in the order they appear in the FileSet. When a match is found, no further clauses are compared and the directory or file is either included or excluded.

The FileSet resource definition below implements this by including specifc directories and files and excluding everything else.

```
FileSet {
  Name = "AllPictures"

  Include {

    File  = "C:/Documents and Settings"

    Options {
      signature = SHA1
      verify = s1
      IgnoreCase = yes

      # Include all users' directories so we reach the inner ones.  Unlike a
      # WildDir pattern ending in *, this RegExDir only matches the top-level
      # directories and not any inner ones.
      RegExDir = "^C:/Documents and Settings/[^/]+$"

      # Ditto all users' My Documents directories.
      WildDir = "C:/Documents and Settings/*/My Documents"

      # Ditto all users' My Documents/My Pictures directories.
      WildDir = "C:/Documents and Settings/*/My Documents/My Pictures"

      # Include the contents of the My Documents/My Pictures directories and
      # any subdirectories.
      Wild = "C:/Documents and Settings/*/My Documents/My Pictures/*"
    }

    Options {
      Exclude = yes
      IgnoreCase = yes

      # Exclude everything else, in particular any files at the top level and
      # any other directories or files in the users' directories.
      Wild = "C:/Documents and Settings/*"
    }
  }
}
```

5.9 Backing up Raw Partitions

The following FileSet definition will backup a raw partition:

```
FileSet {
  Name = "RawPartition"
  Include {
    Options { sparse=yes }
    File = /dev/hda2
  }
}
```

While backing up and restoring a raw partition, you should ensure that no other process including the system is writing to that partition. As a precaution, you are strongly urged to ensure that the raw partition is not mounted or is mounted read-only. If necessary, this can be done using the **RunBeforeJob** directive.

5.10 Excluding Files and Directories

You may also include full filenames or directory names in addition to using wild-cards and **Exclude=yes** in the Options resource as specified above by simply including the files to be excluded in an Exclude resource within the FileSet. For example:

```
FileSet {
  Name = Exclusion_example
  Include {
    Options {
      Signature = SHA1
    }
    File = /
    File = /boot
    File = /home
    File = /rescue
    File = /usr
  }
  Exclude {
    File = /proc
    File = /tmp
    File = .journal
    File = .autofsck
  }
}
```

5.11 Windows FileSets

If you are entering Windows file names, the directory path may be preceded by the drive and a colon (as in c:). However, the path separators must be specified in Unix convention (i.e. forward slash (/)). If you wish to include a quote in a file name, precede the quote with a backslash (\). For example you might use the following for a Windows machine to backup the "My Documents" directory:

```
FileSet {
  Name = "Windows Set"
  Include {
    Options {
       WildFile = "*.obj"
       WildFile = "*.exe"
       exclude = yes
     }
     File = "c:/My Documents"
  }
}
```

For exclude lists to work correctly on Windows, you must observe the following rules:

- Filenames are case sensitive, so you must use the correct case.
- To exclude a directory, you must not have a trailing slash on the directory name.
- If you have spaces in your filename, you must enclose the entire name in double-quote characters ("). Trying to use a backslash before the space will not work.
- If you are using the old Exclude syntax (noted below), you may not specify a drive letter in the exclude. The new syntax noted above should work fine including driver letters.

Thanks to Thiago Lima for summarizing the above items for us. If you are having difficulties getting includes or excludes to work, you might want to try using the **estimate job=xxx listing** command documented in the Console chapter of this manual.

On Win32 systems, if you move a directory or file or rename a file into the set of files being backed up, and a Full backup has already been made, Bacula will not know there are new files to be saved during an Incremental or Differential backup (blame Microsoft, not me). To avoid this problem, please **copy** any new directory or files into the backup area. If you do not have enough disk to copy the directory or files, move them, but then initiate a Full backup.

A Windows Example FileSet The following example was contributed by Russell Howe. Please note that for presentation purposes, the lines beginning with Data and Internet have been wrapped and should included on the previous line with one space.

```
This is my Windows 2000 fileset:
FileSet {
 Name = "Windows 2000"
 Include {
  Options {
   signature = MD5
   Exclude = yes
   IgnoreCase = yes
   # Exclude Mozilla-based programs' file caches
   WildDir = "[A-Z]:/Documents and Settings/*/Application
Data/*/Profiles/*/*/Cache"
   WildDir = "[A-Z]:/Documents and Settings/*/Application
Data/*/Profiles/*/*/Cache.Trash"
   WildDir = "[A-Z]:/Documents and Settings/*/Application
Data/*/Profiles/*/*/ImapMail"

   # Exclude user's registry files - they're always in use anyway.
   WildFile = "[A-Z]:/Documents and Settings/*/Local Settings/Application
Data/Microsoft/Windows/usrclass.*"
   WildFile = "[A-Z]:/Documents and Settings/*/ntuser.*"

   # Exclude directories full of lots and lots of useless little files
   WildDir = "[A-Z]:/Documents and Settings/*/Cookies"
   WildDir = "[A-Z]:/Documents and Settings/*/Recent"
   WildDir = "[A-Z]:/Documents and Settings/*/Local Settings/History"
   WildDir = "[A-Z]:/Documents and Settings/*/Local Settings/Temp"
   WildDir = "[A-Z]:/Documents and Settings/*/Local Settings/Temporary
Internet Files"

   # These are always open and unable to be backed up
   WildFile = "[A-Z]:/Documents and Settings/All Users/Application
Data/Microsoft/Network/Downloader/qmgr[01].dat"

   # Some random bits of Windows we want to ignore
   WildFile = "[A-Z]:/WINNT/security/logs/scepol.log"
   WildDir = "[A-Z]:/WINNT/system32/config"
   WildDir = "[A-Z]:/WINNT/msdownld.tmp"
   WildDir = "[A-Z]:/WINNT/Internet Logs"
   WildDir = "[A-Z]:/WINNT/$Nt*Uninstall*"
   WildDir = "[A-Z]:/WINNT/sysvol"
   WildFile = "[A-Z]:/WINNT/cluster/CLUSDB"
```

```
    WildFile = "[A-Z]:/WINNT/cluster/CLUSDB.LOG"
    WildFile = "[A-Z]:/WINNT/NTDS/edb.log"
    WildFile = "[A-Z]:/WINNT/NTDS/ntds.dit"
    WildFile = "[A-Z]:/WINNT/NTDS/temp.edb"
    WildFile = "[A-Z]:/WINNT/ntfrs/jet/log/edb.log"
    WildFile = "[A-Z]:/WINNT/ntfrs/jet/ntfrs.jdb"
    WildFile = "[A-Z]:/WINNT/ntfrs/jet/temp/tmp.edb"
    WildFile = "[A-Z]:/WINNT/system32/CPL.CFG"
    WildFile = "[A-Z]:/WINNT/system32/dhcp/dhcp.mdb"
    WildFile = "[A-Z]:/WINNT/system32/dhcp/j50.log"
    WildFile = "[A-Z]:/WINNT/system32/dhcp/tmp.edb"
    WildFile = "[A-Z]:/WINNT/system32/LServer/edb.log"
    WildFile = "[A-Z]:/WINNT/system32/LServer/TLSLic.edb"
    WildFile = "[A-Z]:/WINNT/system32/LServer/tmp.edb"
    WildFile = "[A-Z]:/WINNT/system32/wins/j50.log"
    WildFile = "[A-Z]:/WINNT/system32/wins/wins.mdb"
    WildFile = "[A-Z]:/WINNT/system32/wins/winstmp.mdb"

    # Temporary directories & files
    WildDir = "[A-Z]:/WINNT/Temp"
    WildDir = "[A-Z]:/temp"
    WildFile = "*.tmp"
    WildDir = "[A-Z]:/tmp"
    WildDir = "[A-Z]:/var/tmp"

    # Recycle bins
    WildDir = "[A-Z]:/RECYCLER"

    # Swap files
    WildFile = "[A-Z]:/pagefile.sys"

    # These are programs and are easier to reinstall than restore from
    # backup
    WildDir = "[A-Z]:/cygwin"
    WildDir = "[A-Z]:/Program Files/Grisoft"
    WildDir = "[A-Z]:/Program Files/Java"
    WildDir = "[A-Z]:/Program Files/Java Web Start"
    WildDir = "[A-Z]:/Program Files/JavaSoft"
    WildDir = "[A-Z]:/Program Files/Microsoft Office"
    WildDir = "[A-Z]:/Program Files/Mozilla Firefox"
    WildDir = "[A-Z]:/Program Files/Mozilla Thunderbird"
    WildDir = "[A-Z]:/Program Files/mozilla.org"
    WildDir = "[A-Z]:/Program Files/OpenOffice*"
   }

   # Our Win2k boxen all have C: and D: as the main hard drives.
   File = "C:/"
   File = "D:/"
  }
}
```

Note, the three line of the above Exclude were split to fit on the document page, they should be written on a single line in real use.

Windows NTFS Naming Considerations NTFS filenames containing Unicode characters should now be supported as of version 1.37.30 or later.

5.12 Testing Your FileSet

If you wish to get an idea of what your FileSet will really backup or if your exclusion rules will work correctly, you can test it by using the **estimate** command in the Console program. See the estimate in the Console chapter of this manual.

As an example, suppose you add the following test FileSet:

```
FileSet {
```

```
  Name = Test
  Include {
    File = /home/xxx/test
    Options {
       regex = ".*\.c$"
    }
  }
}
```

You could then add some test files to the directory **/home/xxx/test** and use the following command in the console:

```
estimate job=<any-job-name> listing client=<desired-client> fileset=Test
```

to give you a listing of all files that match.

5.13 The Client Resource

The Client resource defines the attributes of the Clients that are served by this Director; that is the machines that are to be backed up. You will need one Client resource definition for each machine to be backed up.

Client (or FileDaemon) Start of the Client directives.

Name = <name> The client name which will be used in the Job resource directive or in the console run command. This directive is required.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address in dotted quad notation for a Bacula File server daemon. This directive is required.

FD Port = <port-number> Where the port is a port number at which the Bacula File server daemon can be contacted. The default is 9102.

Catalog = <Catalog-resource-name> This specifies the name of the catalog resource to be used for this Client. This directive is required.

Password = <password> This is the password to be used when establishing a connection with the File services, so the Client configuration file on the machine to be backed up must have the same password defined for this Director. This directive is required. If you have either **/dev/random bc** on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.

The password is plain text. It is not generated through any special process, but it is preferable for security reasons to make the text random.

File Retention = <time-period-specification> The File Retention directive defines the length of time that Bacula will keep File records in the Catalog database after the End time of the Job corresponding to the File records. When this time period expires, and if **AutoPrune** is set to **yes** Bacula will prune (remove) File records that are older than the specified File Retention period. Note, this affects only records in the catalog database. It does not affect your archive backups.

File records may actually be retained for a shorter period than you specify on this directive if you specify either a shorter **Job Retention** or a shorter **Volume Retention** period. The shortest retention period of the three takes precedence. The time may be expressed in seconds, minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of time specification.

The default is 60 days.

Job Retention = <time-period-specification> The Job Retention directive defines the length of time that Bacula will keep Job records in the Catalog database after the Job End time. When this time period expires, and if **AutoPrune** is set to **yes** Bacula will prune (remove) Job records that are older than the specified File Retention period. As with the other retention periods, this affects only records in the catalog and not data in your archive backup.

If a Job record is selected for pruning, all associated File and JobMedia records will also be pruned regardless of the File Retention period set. As a consequence, you normally will set the File retention period to be less than the Job retention period. The Job retention period can actually be less than the value you specify here if you set the **Volume Retention** directive in the Pool resource to a smaller duration. This is because the Job retention period and the Volume retention period are independently applied, so the smaller of the two takes precedence.

The Job retention period is specified as seconds, minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of time specification.

The default is 180 days.

AutoPrune = <yes|no> If AutoPrune is set to **yes** (default), Bacula (version 1.20 or greater) will automatically apply the File retention period and the Job retention period for the Client at the end of the Job. If you set **AutoPrune = no**, pruning will not be done, and your Catalog will grow in size each time you run a Job. Pruning affects only information in the catalog and not data stored in the backup archives (on Volumes).

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs with the current Client that can run concurrently. Note, this directive limits only Jobs for Clients with the same name as the resource in which it appears. Any other restrictions on the maximum concurrent jobs such as in the Director, Job, or Storage resources will also apply in addition to any limit specified here. The default is set to 1, but you may set it to a larger number.

Priority = <number> The number specifies the priority of this client relative to other clients that the Director is processing simultaneously. The priority can range from 1 to 1000. The clients are ordered such that the smaller number priorities are performed first (not currently implemented).

The following is an example of a valid Client resource definition:

```
Client {
  Name = Minimatou
  FDAddress = minimatou
  Catalog = MySQL
  Password = very_good
}
```

5.14 The Storage Resource

The Storage resource defines which Storage daemons are available for use by the Director.

Storage Start of the Storage resources. At least one storage resource must be specified.

Name = <name> The name of the storage resource. This name appears on the Storage directive specified in the Job resource and is required.

Address = <address> Where the address is a host name, a **fully qualified domain name**, or an **IP address**. Please note that the <address> as specified here will be transmitted to the File daemon who will then use it to contact the Storage daemon. Hence, it is **not**, a good idea to use **localhost** as the name but rather a fully qualified machine name or an IP address. This directive is required.

SD Port = <port> Where port is the port to use to contact the storage daemon for information and to start jobs. This same port number must appear in the Storage resource of the Storage daemon's configuration file. The default is 9103.

Password = <password> This is the password to be used when establishing a connection with the Storage services. This same password also must appear in the Director resource of the Storage daemon's configuration file. This directive is required. If you have either **/dev/random bc** on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.

The password is plain text. It is not generated through any special process, but it is preferable for security reasons to use random text.

Device = <device-name> This directive specifies the Storage daemon's name of the device resource to be used for the storage. If you are using an Autochanger, the name specified here should be the name of the Storage daemon's Autochanger resource rather than the name of an individual device. This name is not the physical device name, but the logical device name as defined on the **Name** directive contained in the **Device** or the **Autochanger** resource definition of the **Storage daemon** configuration file. You can specify any name you would like (even the device name if you prefer) up to a maximum of 127 characters in length. The physical device name associated with this device is specified in the **Storage daemon** configuration file (as **Archive Device**). Please take care not to define two different Storage resource directives in the Director that point to the same Device in the Storage daemon. Doing so may cause the Storage daemon to block (or hang) attempting to open the same device that is already open. This directive is required.

Media Type = <MediaType> This directive specifies the Media Type to be used to store the data. This is an arbitrary string of characters up to 127 maximum that you define. It can be anything you want. However, it is best to make it descriptive of the storage media (e.g. File, DAT, "HP DLT8000", 8mm, ...). In addition, it is essential that you make the **Media Type** specification unique for each storage media type. If you have two DDS-4 drives that have incompatible formats, or if you have a DDS-4 drive and a DDS-4 autochanger, you almost certainly should specify different **Media Types**. During a restore, assuming a **DDS-4** Media Type is associated with the Job, Bacula can decide to use any Storage daemon that supports Media Type **DDS-4** and on any drive that supports it.

If you are writing to disk Volumes, you must make doubly sure that each Device resource defined in the Storage daemon (and hence in the Director's conf file) has a unique media type. Otherwise for Bacula versions 1.38 and older, your restores may not work because Bacula will assume that you can mount any Media Type with the same name on any Device associated with that Media Type. This is possible with tape drives, but with disk drives, unless you are very clever you cannot mount a Volume in any directory – this can be done by creating an appropriate soft link.

Currently Bacula permits only a single Media Type per Storage and Device definition. Consequently, if you have a drive that supports more than one Media Type, you can give a unique string to Volumes with different intrinsic Media Type (Media Type = DDS-3-4 for DDS-3 and DDS-4 types), but then those volumes will only be mounted on drives indicated with the dual type (DDS-3-4).

If you want to tie Bacula to using a single Storage daemon or drive, you must specify a unique Media Type for that drive. This is an important point that should be carefully understood. Note, this applies equally to Disk Volumes. If you define more than one disk Device resource in your Storage daemon's conf file, the Volumes on those two devices are in fact incompatible because one can not be mounted on the other device since they are found in different directories. For this reason, you probably should use two different Media Types for your two disk Devices (even though you might think of them as both being File types). You can find more on this subject in the Basic Volume Management chapter of this manual.

The **MediaType** specified in the Director's Storage resource, **must** correspond to the **Media Type** specified in the **Device** resource of the **Storage daemon** configuration file. This directive is required, and it is used by the Director and the Storage daemon to ensure that a Volume automatically selected from the Pool corresponds to the physical device. If a Storage daemon handles multiple devices (e.g. will write to various file Volumes on different partitions), this directive allows you to specify exactly which device.

As mentioned above, the value specified in the Director's Storage resource must agree with the value specified in the Device resource in the **Storage daemon's** configuration file. It is also an additional check so that you don't try to write data for a DLT onto an 8mm device.

Autochanger = <yes|no> If you specify **yes** for this command (the default is **no**), when you use the **label** command or the **add** command to create a new Volume, **Bacula** will also request the Autochanger Slot number. This simplifies creating database entries for Volumes in an autochanger. If you forget to specify the Slot, the autochanger will not be used. However, you may modify the Slot associated with a Volume at any time by using the **update volume** or **update slots** command in the console program. When **autochanger** is enabled, the algorithm used by Bacula to search for available volumes will be modified to consider only Volumes that are known to be in the autochanger's magazine. If no **in changer** volume is found, Bacula will attempt recycling, pruning, ..., and if still no volume is found, Bacula will search for any volume whether or not in the magazine. By privileging in changer volumes, this procedure minimizes operator intervention. The default is **no**.

For the autochanger to be used, you must also specify **Autochanger = yes** in the Device Resource in the Storage daemon's configuration file as well as other important Storage daemon configuration information. Please consult the Using Autochangers manual of this chapter for the details of using autochangers.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs with the current Storage resource that can run concurrently. Note, this directive limits only Jobs for Jobs using this Storage daemon. Any other restrictions on the maximum concurrent jobs such as in the Director, Job, or Client resources will also apply in addition to any limit specified here. The default is set to 1, but you may set it to a larger number. However, if you set the Storage daemon's number of concurrent jobs greater than one, we recommend that you read the waring documented under Maximum Concurrent Jobs in the Director's resource or simply turn data spooling on as documented in the Data Spooling chapter of this manual.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Director to set a keepalive interval (heartbeat) in seconds on each of the sockets it opens for the Storage resource. This value will override any specified at the Director level. It is implemented only on systems (Linux, ...) that provide the **setsockopt** TCP_KEEPIDLE function. The default value is zero, which means no change is made to the socket.

The following is an example of a valid Storage resource definition:

```
# Definition of tape storage device
Storage {
  Name = DLTDrive
  Address = lpmatou
  Password = storage_password # password for Storage daemon
  Device = "HP DLT 80"    # same as Device in Storage daemon
  Media Type = DLT8000    # same as MediaType in Storage daemon
}
```

5.15 The Pool Resource

The Pool resource defines the set of storage Volumes (tapes or files) to be used by Bacula to write the data. By configuring different Pools, you can determine which set of Volumes (media) receives the backup data. This permits, for example, to store all full backup data on one set of Volumes and all incremental backups on another set of Volumes. Alternatively, you could assign a different set of Volumes to each machine that you backup. This is most easily done by defining multiple Pools.

Another important aspect of a Pool is that it contains the default attributes (Maximum Jobs, Retention Period, Recycle flag, ...) that will be given to a Volume when it is created. This avoids the need for you to answer a large number of questions when labeling a new Volume. Each of these attributes can later be changed on a Volume by Volume basis using the **update** command in the console program. Note that you must explicitly specify which Pool Bacula is to use with each Job. Bacula will not automatically search for the correct Pool.

Most often in Bacula installations all backups for all machines (Clients) go to a single set of Volumes. In this case, you will probably only use the **Default** Pool. If your backup strategy calls for you to mount a different tape each day, you will probably want to define a separate Pool for each day. For more information on this subject, please see the Backup Strategies chapter of this manual.

To use a Pool, there are three distinct steps. First the Pool must be defined in the Director's configuration file. Then the Pool must be written to the Catalog database. This is done automatically by the Director each time that it starts, or alternatively can be done using the **create** command in the console program. Finally, if you change the Pool definition in the Director's configuration file and restart Bacula, the pool will be updated alternatively you can use the **update pool** console command to refresh the database image. It is this database image rather than the Director's resource image that is used for the default Volume attributes. Note, for the pool to be automatically created or updated, it must be explicitly referenced by a Job resource.

Next the physical media must be labeled. The labeling can either be done with the **label** command in the **console** program or using the **btape** program. The preferred method is to use the **label** command in the **console** program.

Finally, you must add Volume names (and their attributes) to the Pool. For Volumes to be used by Bacula they must be of the same **Media Type** as the archive device specified for the job (i.e. if you are going to back up to a DLT device, the Pool must have DLT volumes defined since 8mm volumes cannot be mounted on a DLT drive). The **Media Type** has particular importance if you are backing up to files. When running a Job, you must explicitly specify which Pool to use. Bacula will then automatically select the next Volume to use from the Pool, but it will ensure that the **Media Type** of any Volume selected from the Pool is identical to that required by the Storage resource you have specified for the Job.

If you use the **label** command in the console program to label the Volumes, they will automatically be added to the Pool, so this last step is not normally required.

It is also possible to add Volumes to the database without explicitly labeling the physical volume. This is done with the **add** console command.

As previously mentioned, each time Bacula starts, it scans all the Pools associated with each Catalog, and if the database record does not already exist, it will be created from the Pool Resource definition. **Bacula** probably should do an **update pool** if you change the Pool definition, but currently, you must do this manually using the **update pool** command in the Console program.

The Pool Resource defined in the Director's configuration file (bacula-dir.conf) may contain the following directives:

Pool Start of the Pool resource. There must be at least one Pool resource defined.

Name = <name> The name of the pool. For most applications, you will use the default pool name **Default**. This directive is required.

Maximum Volumes = <number> This directive specifies the maximum number of volumes (tapes or files) contained in the pool. This directive is optional, if omitted or set to zero, any number of volumes will be permitted. In general, this directive is useful for Autochangers where there is a fixed number of Volumes, or for File storage where you wish to ensure that the backups made to disk files do not become too numerous or consume too much space.

Pool Type = <type> This directive defines the pool type, which corresponds to the type of Job being run. It is required and may be one of the following:

Backup

*Archive

*Cloned

*Migration

*Copy

*Save

Note, only Backup is current implemented.

Storage = <storage-resource-name> The Storage directive defines the name of the storage services where you want to backup the FileSet data. For additional details, see the Storage Resource Chapter of this manual. The Storage resource may also be specified in the Job resource, but the value, if any, in the Pool resource overrides any value in the Job. This Storage resource definition is not required by either the Job resource or in the Pool, but it must be specified in one or the other. If not configuration error will result.

Use Volume Once = <yes|no> This directive if set to **yes** specifies that each volume is to be used only once. This is most useful when the Media is a file and you want a new file for each backup that is done. The default is **no** (i.e. use volume any number of times). This directive will most likely be phased out (deprecated), so you are recommended to use **Maximum Volume Jobs = 1** instead.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

Please see the notes below under **Maximum Volume Jobs** concerning using this directive with multiple simultaneous jobs.

Maximum Volume Jobs = <positive-integer> This directive specifies the maximum number of Jobs that can be written to the Volume. If you specify zero (the default), there is no limit. Otherwise, when the number of Jobs backed up to the Volume equals **positive-integer** the Volume will be marked **Used**. When the Volume is marked **Used** it can no longer be used for appending Jobs, much like the **Full** status but it can be recycled if recycling is enabled, and thus used again. By setting **MaximumVolumeJobs** to one, you get the same effect as setting **UseVolumeOnce = yes**.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

If you are running multiple simultaneous jobs, this directive may not work correctly because when a drive is reserved for a job, this directive is not taken into account, so multiple jobs may try to start writing to the Volume. At some point, when the Media record is updated, multiple simultaneous jobs may fail since the Volume can no longer be written.

Maximum Volume Files = <positive-integer> This directive specifies the maximum number of files that can be written to the Volume. If you specify zero (the default), there is no limit. Otherwise, when the number of files written to the Volume equals **positive-integer** the Volume will be marked **Used**. When the Volume is marked **Used** it can no longer be used for appending Jobs, much like the **Full** status but it can be recycled if recycling is enabled and thus used again. This value is checked and the **Used** status is set only at the end of a job that writes to the particular volume.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

Maximum Volume Bytes = <size> This directive specifies the maximum number of bytes that can be written to the Volume. If you specify zero (the default), there is no limit except the physical size of the Volume. Otherwise, when the number of bytes written to the Volume equals **size** the Volume will be marked **Used**. When the Volume is marked **Used** it can no longer be used for appending Jobs, much like the **Full** status but it can be recycled if recycling is enabled, and thus the Volume can be re-used after recycling. This value is checked and the **Used** status set while the job is writing to the particular volume.

This directive is particularly useful for restricting the size of disk volumes, and will work correctly even in the case of multiple simultaneous jobs writing to the volume.

The value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

Volume Use Duration = <time-period-specification> The Volume Use Duration directive defines the time period that the Volume can be written beginning from the time of first data write to the Volume. If the time-period specified is zero (the default), the Volume can be written indefinitely. Otherwise, the next time a job runs that wants to access this Volume, and the time period from the first write to the volume (the first Job written) exceeds the time-period-specification, the Volume will be marked **Used**, which means that no more Jobs can be appended to the Volume, but it may be recycled if recycling is enabled. Using the command **status dir** applies algorithms similar to running jobs, so during such a command, the Volume status may also be changed. Once the Volume is recycled, it will be available for use again.

You might use this directive, for example, if you have a Volume used for Incremental backups, and Volumes used for Weekly Full backups. Once the Full backup is done, you will want to use a different Incremental Volume. This can be accomplished by setting the Volume Use Duration for the Incremental Volume to six days. I.e. it will be used for the 6 days following a Full save, then a different Incremental volume will be used. Be careful about setting the duration to short periods such as 23 hours, or you might experience problems of Bacula waiting for a tape over the weekend only to complete the backups Monday morning when an operator mounts a new tape.

The use duration is checked and the **Used** status is set only at the end of a job that writes to the particular volume, which means that even though the use duration may have expired, the catalog entry will not be updated until the next job that uses this volume is run. This directive is not intended to be used to limit volume sizes and will not work correctly (i.e. will fail jobs) if the use duration expires while multiple simultaneous jobs are writing to the volume.

Please note that the value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update volume** command in the Console.

Catalog Files = <yes|no> This directive defines whether or not you want the names of the files that were saved to be put into the catalog. The default is **yes**. The advantage of specifying **Catalog Files = No** is that you will have a significantly smaller Catalog database. The disadvantage is that you will not be able to produce a Catalog listing of the files backed up for each Job (this is often called Browsing). Also, without the File entries in the catalog, you will not be able to use the Console **restore** command nor any other command that references File entries.

AutoPrune = <yes|no> If AutoPrune is set to **yes** (default), Bacula (version 1.20 or greater) will automatically apply the Volume Retention period when new Volume is needed and no appendable Volumes exist in the Pool. Volume pruning causes expired Jobs (older than the **Volume Retention** period) to be deleted from the Catalog and permits possible recycling of the Volume.

Volume Retention = <time-period-specification> The Volume Retention directive defines the length of time that **Bacula** will keep records associated with the Volume in the Catalog database after the End time of each Job written to the Volume. When this time period expires, and if **AutoPrune** is set to **yes** Bacula may prune (remove) Job records that are older than the specified Volume Retention period if it is necessary to free up a Volume. Recycling will not occur until it is absolutely necessary to free up a volume (i.e. no other writable volume exists). All File records associated with pruned Jobs are also pruned. The time may be specified as seconds, minutes, hours, days, weeks, months, quarters, or years. The **Volume Retention** is applied independently of the **Job Retention** and the **File Retention** periods defined in the Client resource. This means that all the retentions periods are applied in turn and that the shorter period is the one that effectively takes precedence. Note, that when the **Volume Retention** period has been reached, and it is necessary to obtain a new volume, Bacula will prune both the Job and the File records. This pruning could also occur during a **status dir** command because it uses similar algorithms for finding the next available Volume.

It is important to know that when the Volume Retention period expires, Bacula does not automatically recycle a Volume. It attempts to keep the Volume data intact as long as possible before over writing the Volume.

By defining multiple Pools with different Volume Retention periods, you may effectively have a set of tapes that is recycled weekly, another Pool of tapes that is recycled monthly and so on. However, one must keep in mind that if your **Volume Retention** period is too short, it may prune the last valid Full backup, and hence until the next Full backup is done, you will not have a complete backup of your system, and in addition, the next Incremental or Differential backup will be promoted to a Full backup. As a consequence, the minimum **Volume Retention** period should be at twice the interval of your Full backups. This means that if you do a Full backup once a month, the minimum Volume retention period should be two months.

The default Volume retention period is 365 days, and either the default or the value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

ScratchPool = <pool-resource-name> This directive permits to specify a dedicate *Scratch* for the current pool. This pool will replace the special pool named *Scrach* for volume selection. For more information about *Scratch* see Scratch Pool section of this manual. This is useful when using multiple storage sharing the same mediatype or when you want to dedicate volumes to a particular set of pool.

RecyclePool = <pool-resource-name> This directive defines to which pool the Volume will be placed (moved) when it is recycled. Without this directive, a Volume will remain in the same pool when it is recycled. With this directive, it can be moved automatically to any existing pool during a recycle. This directive is probably most useful when defined in the Scratch pool, so that volumes will be recycled back into the Scratch pool. For more on the see the Scratch Pool section of this manual.

Although this directive is called RecyclePool, the Volume in question is actually moved from its current pool to the one you specify on this directive when Bacula prunes the Volume and discovers that there are no records left in the catalog and hence marks it as **Purged**.

Recycle = <yes|no> This directive specifies whether or not Purged Volumes may be recycled. If it is set to **yes** (default) and Bacula needs a volume but finds none that are appendable, it will search for and recycle (reuse) Purged Volumes (i.e. volumes with all the Jobs and Files expired and thus deleted from the Catalog). If the Volume is recycled, all previous data written to that Volume will be overwritten. If Recycle is set to **no**, the Volume will not be recycled, and hence, the data will remain valid. If you want to reuse (re-write) the Volume, and the recycle flag is no (0 in the catalog), you may manually set the recycle flag (update command) for a Volume to be reused.

Please note that the value defined by this directive in the bacula-dir.conf file is the default value used when a Volume is created. Once the volume is created, changing the value in the bacula-dir.conf file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

When all Job and File records have been pruned or purged from the catalog for a particular Volume, if that Volume is marked as Append, Full, Used, or Error, it will then be marked as Purged. Only Volumes marked as Purged will be considered to be converted to the Recycled state if the **Recycle** directive is set to **yes**.

Recycle Oldest Volume = <yes|no> This directive instructs the Director to search for the oldest used Volume in the Pool when another Volume is requested by the Storage daemon and none are available. The catalog is then **pruned** respecting the retention periods of all Files and Jobs written to this Volume. If all Jobs are pruned (i.e. the volume is Purged), then the Volume is recycled and will be used as the next Volume to be written. This directive respects any Job, File, or Volume retention periods that you may have specified, and as such it is **much** better to use this directive than the Purge Oldest Volume.

This directive can be useful if you have a fixed number of Volumes in the Pool and you want to cycle through them and you have specified the correct retention periods.

However, if you use this directive and have only one Volume in the Pool, you will immediately recycle your Volume if you fill it and Bacula needs another one. Thus your backup will be totally invalid. Please use this directive with care. The default is **no**.

Recycle Current Volume = <yes|no> If Bacula needs a new Volume, this directive instructs Bacula to Prune the volume respecting the Job and File retention periods. If all Jobs are pruned (i.e. the volume is Purged), then the Volume is recycled and will be used as the next Volume to be written. This directive respects any Job, File, or Volume retention periods that you may have specified, and thus it is **much** better to use it rather than the Purge Oldest Volume directive.

This directive can be useful if you have: a fixed number of Volumes in the Pool, you want to cycle through them, and you have specified retention periods that prune Volumes before you have cycled through the Volume in the Pool.

However, if you use this directive and have only one Volume in the Pool, you will immediately recycle your Volume if you fill it and Bacula needs another one. Thus your backup will be totally invalid. Please use this directive with care. The default is **no**.

Purge Oldest Volume = <yes|no> This directive instructs the Director to search for the oldest used Volume in the Pool when another Volume is requested by the Storage daemon and none are available. The catalog is then **purged** irrespective of retention periods of all Files and Jobs written to this Volume. The Volume is then recycled and will be used as the next Volume to be written. This directive overrides any Job, File, or Volume retention periods that you may have specified.

This directive can be useful if you have a fixed number of Volumes in the Pool and you want to cycle through them and reusing the oldest one when all Volumes are full, but you don't want to worry about setting proper retention periods. However, by using this option you risk losing valuable data.

Please be aware that **Purge Oldest Volume** disregards all retention periods. If you have only a single Volume defined and you turn this variable on, that Volume will always be immediately overwritten when it fills! So at a minimum, ensure that you have a decent number of Volumes in your Pool before running any jobs. If you want retention periods to apply do not use this directive. To specify a retention period, use the **Volume Retention** directive (see above).

We **highly** recommend against using this directive, because it is sure that some day, Bacula will recycle a Volume that contains current data. The default is **no**.

Cleaning Prefix = <string> This directive defines a prefix string, which if it matches the beginning of a Volume name during labeling of a Volume, the Volume will be defined with the VolStatus set to **Cleaning** and thus Bacula will never attempt to use this tape. This is primarily for use with autochangers that accept barcodes where the convention is that barcodes beginning with **CLN** are treated as cleaning tapes.

Label Format = <format> This directive specifies the format of the labels contained in this pool. The format directive is used as a sort of template to create new Volume names during automatic Volume labeling.

The **format** should be specified in double quotes, and consists of letters, numbers and the special characters hyphen (**-**), underscore (_), colon (**:**), and period (**.**), which are the legal characters for a Volume name. The **format** should be enclosed in double quotes (").

In addition, the format may contain a number of variable expansion characters which will be expanded by a complex algorithm allowing you to create Volume names of many different formats. In all cases, the expansion process must resolve to the set of characters noted above that are legal Volume names. Generally, these variable expansion characters begin with a dollar sign (**$**) or a left bracket ([). If you specify variable expansion characters, you should always enclose the format with double quote characters (**"**). For more details on variable expansion, please see the Variable Expansion Chapter of this manual.

If no variable expansion characters are found in the string, the Volume name will be formed from the **format** string appended with the a unique number that increases. If you do not remove volumes from the pool, this number should be the number of volumes plus one, but this is not guaranteed. The unique number will be edited as four digits with leading zeros. For example, with a **Label Format = "File-"**, the first volumes will be named **File-0001**, **File-0002**, ...

With the exception of Job specific variables, you can test your **LabelFormat** by using the var command the Console Chapter of this manual.

In almost all cases, you should enclose the format specification (part after the equal sign) in double quotes. Please note that this directive is deprecated and is replaced in version 1.37 and greater with a Python script for creating volume names.

In order for a Pool to be used during a Backup Job, the Pool must have at least one Volume associated with it. Volumes are created for a Pool using the **label** or the **add** commands in the **Bacula Console**, program. In addition to adding Volumes to the Pool (i.e. putting the Volume names in the Catalog database), the physical Volume must be labeled with a valid Bacula software volume label before **Bacula** will accept the Volume. This will be automatically done if you use the **label** command. Bacula can automatically label Volumes if instructed to do so, but this feature is not yet fully implemented.

The following is an example of a valid Pool resource definition:

```
Pool {
  Name = Default
  Pool Type = Backup
}
```

5.15.1 The Scratch Pool

In general, you can give your Pools any name you wish, but there is one important restriction: the Pool named **Scratch**, if it exists behaves like a scratch pool of Volumes in that when Bacula needs a new Volume for writing and it cannot find one, it will look in the Scratch pool, and if it finds an available Volume, it will move it out of the Scratch pool into the Pool currently being used by the job.

5.16 The Catalog Resource

The Catalog Resource defines what catalog to use for the current job. Currently, Bacula can only handle a single database server (SQLite, MySQL, PostgreSQL) that is defined when configuring **Bacula**. However, there may be as many Catalogs (databases) defined as you wish. For example, you may want each Client to have its own Catalog database, or you may want backup jobs to use one database and verify or restore jobs to use another database.

Since SQLite is compiled in, it always runs on the same machine as the Director and the database must be directly accessible (mounted) from the Director. However, since both MySQL and PostgreSQL are networked databases, they may reside either on the same machine as the Director or on a different machine on the network. See below for more details.

Catalog Start of the Catalog resource. At least one Catalog resource must be defined.

Name = <name> The name of the Catalog. No necessary relation to the database server name. This name will be specified in the Client resource directive indicating that all catalog data for that Client is maintained in this Catalog. This directive is required.

password = <password> This specifies the password to use when logging into the database. This directive is required.

DB Name = <name> This specifies the name of the database. If you use multiple catalogs (databases), you specify which one here. If you are using an external database server rather than the internal one, you must specify a name that is known to the server (i.e. you explicitly created the Bacula tables using this name. This directive is required.

user = <user> This specifies what user name to use to log into the database. This directive is required.

DB Socket = <socket-name> This is the name of a socket to use on the local host to connect to the database. This directive is used only by MySQL and is ignored by SQLite. Normally, if neither **DB Socket** or **DB Address** are specified, MySQL will use the default socket. If the DB Socket is specified, the MySQL server must reside on the same machine as the Director.

DB Address = <address> This is the host address of the database server. Normally, you would specify this instead of **DB Socket** if the database server is on another machine. In that case, you will also specify **DB Port**. This directive is used only by MySQL and PostgreSQL and is ignored by SQLite if provided. This directive is optional.

DB Port = <port> This defines the port to be used in conjunction with **DB Address** to access the database if it is on another machine. This directive is used only by MySQL and PostgreSQL and is ignored by SQLite if provided. This directive is optional.

the different

The following is an example of a valid Catalog resource definition:

```
Catalog
{
  Name = SQLite
  dbname = bacula;
  user = bacula;
  password = ""                       # no password = no security
}
```

or for a Catalog on another machine:

```
Catalog
{
  Name = MySQL
  dbname = bacula
  user = bacula
  password = ""
  DB Address = remote.acme.com
  DB Port = 1234
}
```

5.17 The Messages Resource

For the details of the Messages Resource, please see the Messages Resource Chapter of this manual.

5.18 The Console Resource

As of Bacula version 1.33 and higher, there are three different kinds of consoles, which the administrator or user can use to interact with the Director. These three kinds of consoles comprise three different security levels.

- The first console type is an **anonymous** or **default** console, which has full privileges. There is no console resource necessary for this type since the password is specified in the Director's resource and consequently such consoles do not have a name as defined on a **Name =** directive. This is the kind of console that was initially implemented in versions prior to 1.33 and remains valid. Typically you would use it only for administrators.
- The second type of console, and new to version 1.33 and higher is a "named" console defined within a Console resource in both the Director's configuration file and in the Console's configuration file. Both the names and the passwords in these two entries must match much as is the case for Client programs.

  This second type of console begins with absolutely no privileges except those explicitly specified in the Director's Console resource. Thus you can have multiple Consoles with different names and passwords, sort of like multiple users, each with different privileges. As a default, these consoles can do absolutely nothing – no commands whatsoever. You give them privileges or rather access to commands and resources by specifying access control lists in the Director's Console resource. The ACLs are specified by a directive followed by a list of access names. Examples of this are shown below.
- The third type of console is similar to the above mentioned one in that it requires a Console resource definition in both the Director and the Console. In addition, if the console name, provided on the **Name =** directive, is the same as a Client name, that console is permitted to use the **SetIP** command to change the Address directive in the Director's client resource to the IP address of the Console. This permits portables or other machines using DHCP (non-fixed IP addresses) to "notify" the Director of their current IP address.

The Console resource is optional and need not be specified. The following directives are permitted within the Director's configuration resource:

Name = <name> The name of the console. This name must match the name specified in the Console's configuration resource (much as is the case with Client definitions).

Password = <password> Specifies the password that must be supplied for a named Bacula Console to be authorized. The same password must appear in the **Console** resource of the Console configuration file. For added security, the password is never actually passed across the network but rather a challenge response hash code created with the password. This directive is required. If you have either

/dev/random bc on your machine, Bacula will generate a random password during the configuration process, otherwise it will be left blank.

The password is plain text. It is not generated through any special process. However, it is preferable for security reasons to choose random text.

JobACL = <name-list> This directive is used to specify a list of Job resource names that can be accessed by the console. Without this directive, the console cannot access any of the Director's Job resources. Multiple Job resource names may be specified by separating them with commas, and/or by specifying multiple JobACL directives. For example, the directive may be specified as:

```
JobACL = kernsave, "Backup client 1", "Backup client 2"
JobACL = "RestoreFiles"
```

With the above specification, the console can access the Director's resources for the four jobs named on the JobACL directives, but for no others.

ClientACL = <name-list> This directive is used to specify a list of Client resource names that can be accessed by the console.

StorageACL = <name-list> This directive is used to specify a list of Storage resource names that can be accessed by the console.

ScheduleACL = <name-list> This directive is used to specify a list of Schedule resource names that can be accessed by the console.

PoolACL = <name-list> This directive is used to specify a list of Pool resource names that can be accessed by the console.

FileSetACL = <name-list> This directive is used to specify a list of FileSet resource names that can be accessed by the console.

CatalogACL = <name-list> This directive is used to specify a list of Catalog resource names that can be accessed by the console.

CommandACL = <name-list> This directive is used to specify a list of of console commands that can be executed by the console.

WhereACL = <string> This directive permits you to specify where a restricted console can restore files. If this directive is not specified, only the default restore location is permitted (normally **/tmp/bacula-restores**. If ***all*** is specified any path the user enters will be accepted (not very secure), any other value specified (there may be multiple WhereACL directives) will restrict the user to use that path. For example, on a Unix system, if you specify "/", the file will be restored to the original location. This directive is untested.

Aside from Director resource names and console command names, the special keyword ***all*** can be specified in any of the above access control lists. When this keyword is present, any resource or command name (which ever is appropriate) will be accepted. For an example configuration file, please see the Console Configuration chapter of this manual.

5.19 The Counter Resource

The Counter Resource defines a counter variable that can be accessed by variable expansion used for creating Volume labels with the **LabelFormat** directive. See the LabelFormat directive in this chapter for more details.

Counter Start of the Counter resource. Counter directives are optional.

Name = <name> The name of the Counter. This is the name you will use in the variable expansion to reference the counter value.

Minimum = <integer> This specifies the minimum value that the counter can have. It also becomes the default. If not supplied, zero is assumed.

Maximum = <integer> This is the maximum value value that the counter can have. If not specified or set to zero, the counter can have a maximum value of 2,147,483,648 (2 to the 31 power). When the counter is incremented past this value, it is reset to the Minimum.

***WrapCounter = <counter-name>** If this value is specified, when the counter is incremented past the maximum and thus reset to the minimum, the counter specified on the **WrapCounter** is incremented. (This is not currently implemented).

Catalog = <catalog-name> If this directive is specified, the counter and its values will be saved in the specified catalog. If this directive is not present, the counter will be redefined each time that Bacula is started.

5.20 Example Director Configuration File

An example Director configuration file might be the following:

```
#
# Default Bacula Director Configuration file
#
#  The only thing that MUST be changed is to add one or more
#   file or directory names in the Include directive of the
#   FileSet resource.
#
#  For Bacula release 1.15 (5 March 2002) -- redhat
#
#  You might also want to change the default email address
#   from root to your address.  See the "mail" and "operator"
#   directives in the Messages resource.
#
Director {                            # define myself
  Name = rufus-dir
  QueryFile = "/home/kern/bacula/bin/query.sql"
  WorkingDirectory = "/home/kern/bacula/bin/working"
  PidDirectory = "/home/kern/bacula/bin/working"
  Password = "XkSfzu/Cf/wX4L8Zh4G4/yhCbpLcz3YVdmVoQvU3EyF/"
}
# Define the backup Job
Job {
  Name = "NightlySave"
  Type = Backup
  Level = Incremental                 # default
  Client=rufus-fd
  FileSet="Full Set"
  Schedule = "WeeklyCycle"
  Storage = DLTDrive
  Messages = Standard
  Pool = Default
}
Job {
  Name = "Restore"
  Type = Restore
  Client=rufus-fd
  FileSet="Full Set"
  Where = /tmp/bacula-restores
  Storage = DLTDrive
  Messages = Standard
  Pool = Default
}

# List of files to be backed up
FileSet {
  Name = "Full Set"
  Include {
    Options { signature=SHA1}
#
#  Put your list of files here, one per line or include an
```

```
#    external list with:
#
#    @file-name
#
#  Note: / backs up everything
  File = /
}
  Exclude {}
}
# When to do the backups
Schedule {
  Name = "WeeklyCycle"
  Run = level=Full sun at 2:05
  Run = level=Incremental mon-sat at 2:05
}
# Client (File Services) to backup
Client {
  Name = rufus-fd
  Address = rufus
  Catalog = MyCatalog
  Password = "MQk6lVinz4GG2hdIZk1dsKE/LxMZGo6znMHiD7t7vzF+"
  File Retention = 60d      # sixty day file retention
  Job Retention = 1y        # 1 year Job retention
  AutoPrune = yes           # Auto apply retention periods
}
# Definition of DLT tape storage device
Storage {
  Name = DLTDrive
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = "HP DLT 80"      # same as Device in Storage daemon
  Media Type = DLT8000      # same as MediaType in Storage daemon
}
# Definition for a DLT autochanger device
Storage {
  Name = Autochanger
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = "Autochanger"    # same as Device in Storage daemon
  Media Type = DLT-8000     # Different from DLTDrive
  Autochanger = yes
}
# Definition of DDS tape storage device
Storage {
  Name = SDT-10000
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = SDT-10000        # same as Device in Storage daemon
  Media Type = DDS-4        # same as MediaType in Storage daemon
}
# Definition of 8mm tape storage device
Storage {
  Name = "8mmDrive"
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = "Exabyte 8mm"
  MediaType = "8mm"
}
# Definition of file storage device
Storage {
  Name = File
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = FileStorage
  Media Type = File
}
# Generic catalog service
Catalog {
  Name = MyCatalog
  dbname = bacula; user = bacula; password = ""
}
# Reasonable message delivery -- send most everything to
#   the email address and to the console
Messages {
  Name = Standard
  mail = root@localhost = all, !skipped, !terminate
```

```
  operator = root@localhost = mount
  console = all, !skipped, !saved
}

# Default pool definition
Pool {
  Name = Default
  Pool Type = Backup
  AutoPrune = yes
  Recycle = yes
}
#
# Restricted console used by tray-monitor to get the status of the director
#
Console {
  Name = Monitor
  Password = "GN0uRo7PTUmlMbqrJ2Gr1p0fk0HQJTxwnFyE4WSST3MWZseR"
  CommandACL = status, .status
}
```

Chapter 6

Client/File daemon Configuration

The Client (or File Daemon) Configuration is one of the simpler ones to specify. Generally, other than changing the Client name so that error messages are easily identified, you will not need to modify the default Client configuration file.

For a general discussion of configuration file and resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual. The following Client Resource definitions must be defined:

- Client – to define what Clients are to be backed up.
- Director – to define the Director's name and its access password.
- Messages – to define where error and information messages are to be sent.

6.1 The Client Resource

The Client Resource (or FileDaemon) resource defines the name of the Client (as used by the Director) as well as the port on which the Client listens for Director connections.

Client (or FileDaemon) Start of the Client records. There must be one and only one Client resource in the configuration file, since it defines the properties of the current client program.

Name = <name> The client name that must be used by the Director when connecting. Generally, it is a good idea to use a name related to the machine so that error messages can be easily identified if you have multiple Clients. This directive is required.

Working Directory = <Directory> This directive is mandatory and specifies a directory in which the File daemon may put its status files. This directory should be used only by **Bacula**, but may be shared by other Bacula daemons provided the daemon names on the **Name** definition are unique for each daemon. This directive is required.

On Win32 systems, in some circumstances you may need to specify a drive letter in the specified working directory path. Also, please be sure that this directory is writable by the SYSTEM user otherwise restores may fail (the bootstrap file that is transferred to the File daemon from the Director is temporarily put in this directory before being passed to the Storage daemon).

Pid Directory = <Directory> This directive is mandatory and specifies a directory in which the Director may put its process Id file files. The process Id file is used to shutdown Bacula and to prevent multiple copies of Bacula from running simultaneously. This record is required. Standard shell expansion of the **Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

Typically on Linux systems, you will set this to: **/var/run**. If you are not installing Bacula in the system directories, you can use the **Working Directory** as defined above.

Heartbeat Interval = <time-interval> This record defines an interval of time in seconds. For each heartbeat that the File daemon receives from the Storage daemon, it will forward it to the Director. In addition, if no heartbeat has been received from the Storage daemon and thus forwarded the File daemon will send a heartbeat signal to the Director and to the Storage daemon to keep the channels active. The default interval is zero which disables the heartbeat. This feature is particularly useful if you have a router such as 3Com that does not follow Internet standards and times out a valid connection after a short duration despite the fact that keepalive is set. This usually results in a broken pipe error message.

If you continue getting broken pipe error messages despite using the Heartbeat Interval, and you are using Windows, you should consider upgrading your ethernet driver. This is a known problem with NVidia NForce 3 drivers (4.4.2 17/05/2004), or try the following workaround suggested by Thomas Simmons for Win32 machines:

Browse to: Start > Control Panel > Network Connections

Right click the connection for the nvidia adapter and select properties. Under the General tab, click "Configure...". Under the Advanced tab set "Checksum Offload" to disabled and click OK to save the change.

Lack of communications, or communications that get interrupted can also be caused by Linux firewalls where you have a rule that throttles connections or traffic.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs that should run concurrently. The default is set to 2, but you may set it to a larger number. Each contact from the Director (e.g. status request, job start request) is considered as a Job, so if you want to be able to do a **status** request in the console at the same time as a Job is running, you will need to set this value greater than 1.

FDAddresses = <IP-address-specification> Specify the ports and addresses on which the File daemon listens for Director connections. Probably the simplest way to explain is to show an example:

```
FDAddresses  = {
   ip = { addr = 1.2.3.4; port = 1205; }
   ipv4 = {
       addr = 1.2.3.4; port = http; }
   ipv6 = {
       addr = 1.2.3.4;
       port = 1205;
   }
   ip = {
       addr = 1.2.3.4
       port = 1205
   }
   ip = { addr = 1.2.3.4 }
   ip = {
       addr = 201:220:222::2
   }
   ip = {
       addr = bluedot.thun.net
   }
}
```

where ip, ip4, ip6, addr, and port are all keywords. Note, that the address can be specified as either a dotted quadruple, or IPv6 colon notation, or as a symbolic name (only in the ip specification). Also, port can be specified as a number or as the mnemonic value from the /etc/services file. If a port is not specified, the default will be used. If an ip section is specified, the resolution can be made either by IPv4 or IPv6. If ip4 is specified, then only IPv4 resolutions will be permitted, and likewise with ip6.

FDPort = <port-number> This specifies the port number on which the Client listens for Director connections. It must agree with the FDPort specified in the Client resource of the Director's configuration file. The default is 9102.

FDAddress = <IP-Address> This record is optional, and if it is specified, it will cause the File daemon server (for Director connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this record is not specified, the File daemon will bind to any available address (the default).

FDSourceAddress = <IP-Address> This record is optional, and if it is specified, it will cause the File daemon server (for Storage connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this record is not specified, the kernel will choose the best address according to the routing table (the default).

SDConnectTimeout = <time-interval> This record defines an interval of time that the File daemon will try to connect to the Storage daemon. The default is 30 minutes. If no connection is made in the specified time interval, the File daemon cancels the Job.

Maximum Network Buffer Size = <bytes> where <bytes> specifies the initial network buffer size to use with the File daemon. This size will be adjusted down if it is too large until it is accepted by the OS. Please use care in setting this value since if it is too large, it will be trimmed by 512 bytes until the OS is happy, which may require a large number of system calls. The default value is 65,536 bytes.

Note, on certain Windows machines, there are reports that the transfer rates are very slow and this seems to be related to the default 65,536 size. On systems where the transfer rates seem abnormally slow compared to other systems, you might try setting the Maximum Network Buffer Size to 32,768 in both the File daemon and in the Storage daemon.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the File daemon to set a keepalive interval (heartbeat) in seconds on each of the sockets to communicate with the Storage daemon. It is implemented only on systems (Linux, ...) that provide the **setsockopt** TCP_KEEPIDLE function. The default value is zero, which means no change is made to the socket.

PKI Encryption See the Data Encryption chapter of this manual.

PKI Signatures See the Data Encryption chapter of this manual.

PKI Keypair See the Data Encryption chapter of this manual.

PKI Master Key See the Data Encryption chapter of this manual.

The following is an example of a valid Client resource definition:

```
Client {                              # this is me
  Name = rufus-fd
  WorkingDirectory = $HOME/bacula/bin/working
  Pid Directory = $HOME/bacula/bin/working
}
```

6.2 The Director Resource

The Director resource defines the name and password of the Directors that are permitted to contact this Client.

Director Start of the Director records. There may be any number of Director resources in the Client configuration file. Each one specifies a Director that is allowed to connect to this Client.

Name = <name> The name of the Director that may contact this Client. This name must be the same as the name specified on the Director resource in the Director's configuration file. Note, the case (upper/lower) of the characters in the name are significant (i.e. S is not the same as s). This directive is required.

Password = <password> Specifies the password that must be supplied for a Director to be authorized. This password must be the same as the password specified in the Client resource in the Director's configuration file. This directive is required.

Monitor = <yes|no> If Monitor is set to **no** (default), this director will have full access to this Client. If Monitor is set to **yes**, this director will only be able to fetch the current status of this Client.

Please note that if this director is being used by a Monitor, we highly recommend to set this directive to **yes** to avoid serious security problems.

Thus multiple Directors may be authorized to use this Client's services. Each Director will have a different name, and normally a different password as well.

The following is an example of a valid Director resource definition:

```
#
# List Directors who are permitted to contact the File daemon
#
Director {
  Name = HeadMan
  Password = very_good                # password HeadMan must supply
}
Director {
  Name = Worker
  Password = not_as_good
  Monitor = Yes
}
```

6.3 The Message Resource

Please see the Messages Resource Chapter of this manual for the details of the Messages Resource.

There must be at least one Message resource in the Client configuration file.

6.4 Example Client Configuration File

An example File Daemon configuration file might be the following:

```
#
# Default  Bacula File Daemon Configuration file
#
#  For Bacula release 1.35.2 (16 August 2004) -- gentoo 1.4.16
#
# There is not much to change here except perhaps to
#   set the Director's name and File daemon's name
#   to something more appropriate for your site.
#
#
# List Directors who are permitted to contact this File daemon
#
Director {
  Name = rufus-dir
  Password = "/LqPRkX++saVyQE7w7mmiFg/qxYc1kufww6FEyY/47jU"
}
#
# Restricted Director, used by tray-monitor to get the
#   status of the file daemon
#
Director {
  Name = rufus-mon
  Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
  Monitor = yes
}
#
# "Global" File daemon configuration specifications
#
FileDaemon {                          # this is me
  Name = rufus-fd
  WorkingDirectory = $HOME/bacula/bin/working
  Pid Directory = $HOME/bacula/bin/working
}
# Send all messages except skipped files back to Director
Messages {
  Name = Standard
  director = rufus-dir = all, !skipped
}
```

Chapter 7

Storage Daemon Configuration

The Storage Daemon configuration file has relatively few resource definitions. However, due to the great variation in backup media and system capabilities, the storage daemon must be highly configurable. As a consequence, there are quite a large number of directives in the Device Resource definition that allow you to define all the characteristics of your Storage device (normally a tape drive). Fortunately, with modern storage devices, the defaults are sufficient, and very few directives are actually needed.

Examples of **Device** resource directives that are known to work for a number of common tape drives can be found in the **<bacula-src>/examples/devices** directory, and most will also be listed here.

For a general discussion of configuration file and resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual. The following Storage Resource definitions must be defined:

- Storage – to define the name of the Storage daemon.
- Director – to define the Director's name and his access password.
- Device – to define the characteristics of your storage device (tape drive).
- Messages – to define where error and information messages are to be sent.

7.1 Storage Resource

In general, the properties specified under the Storage resource define global properties of the Storage daemon. Each Storage daemon configuration file must have one and only one Storage resource definition.

Name = <Storage-Daemon-Name> Specifies the Name of the Storage daemon. This directive is required.

Working Directory = <Directory> This directive is mandatory and specifies a directory in which the Storage daemon may put its status files. This directory should be used only by **Bacula**, but may be shared by other Bacula daemons provided the names given to each daemon are unique. This directive is required

Pid Directory = <Directory> This directive is mandatory and specifies a directory in which the Director may put its process Id file files. The process Id file is used to shutdown Bacula and to prevent multiple copies of Bacula from running simultaneously. This directive is required. Standard shell expansion of the **Directory** is done when the configuration file is read so that values such as **$HOME** will be properly expanded.

Typically on Linux systems, you will set this to: **/var/run**. If you are not installing Bacula in the system directories, you can use the **Working Directory** as defined above.

Heartbeat Interval = <time-interval> This directive defines an interval of time in seconds. When the Storage daemon is waiting for the operator to mount a tape, each time interval, it will send a heartbeat signal to the File daemon. The default interval is zero which disables the heartbeat. This feature is particularly useful if you have a router such as 3Com that does not follow Internet standards and times out an valid connection after a short duration despite the fact that keepalive is set. This usually results in a broken pipe error message.

Client Connect Wait = <time-interval> This directive defines an interval of time in seconds that the Storage daemon will wait for a Client (the File daemon) to connect. The default is 30 minutes. Be aware that the longer the Storage daemon waits for a Client, the more resources will be tied up.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs that may run concurrently. The default is set to 10, but you may set it to a larger number. Each contact from the Director (e.g. status request, job start request) is considered as a Job, so if you want to be able to do a **status** request in the console at the same time as a Job is running, you will need to set this value greater than 1. To run simultaneous Jobs, you will need to set a number of other directives in the Director's configuration file. Which ones you set depend on what you want, but you will almost certainly need to set the **Maximum Concurrent Jobs** in the Storage resource in the Director's configuration file and possibly those in the Job and Client resources.

SDAddresses = <IP-address-specification> Specify the ports and addresses on which the Storage daemon will listen for Director connections. Normally, the default is sufficient and you do not need to specify this directive. Probably the simplest way to explain how this directive works is to show an example:

```
 SDAddresses  = { ip = {
        addr = 1.2.3.4; port = 1205; }
    ipv4 = {
        addr = 1.2.3.4; port = http; }
    ipv6 = {
        addr = 1.2.3.4;
        port = 1205;
    }
    ip = {
        addr = 1.2.3.4
        port = 1205
    }
    ip = {
        addr = 1.2.3.4
    }
    ip = {
        addr = 201:220:222::2
    }
    ip = {
        addr = bluedot.thun.net
    }
}
```

where ip, ip4, ip6, addr, and port are all keywords. Note, that the address can be specified as either a dotted quadruple, or IPv6 colon notation, or as a symbolic name (only in the ip specification). Also, port can be specified as a number or as the mnemonic value from the /etc/services file. If a port is not specified, the default will be used. If an ip section is specified, the resolution can be made either by IPv4 or IPv6. If ip4 is specified, then only IPv4 resolutions will be permitted, and likewise with ip6.

Using this directive, you can replace both the SDPort and SDAddress directives shown below.

SDPort = <port-number> Specifies port number on which the Storage daemon listens for Director connections. The default is 9103.

SDAddress = <IP-Address> This directive is optional, and if it is specified, it will cause the Storage daemon server (for Director and File daemon connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this directive is not specified, the Storage daemon will bind to any available address (the default).

The following is a typical Storage daemon Storage definition.

```
#
# "Global" Storage daemon configuration specifications appear
# under the Storage resource.
#
Storage {
  Name = "Storage daemon"
  Address = localhost
  WorkingDirectory = "~/bacula/working"
  Pid    Directory = "~/bacula/working"
}
```

7.2 Director Resource

The Director resource specifies the Name of the Director which is permitted to use the services of the Storage daemon. There may be multiple Director resources. The Director Name and Password must match the corresponding values in the Director's configuration file.

Name = <Director-Name> Specifies the Name of the Director allowed to connect to the Storage daemon. This directive is required.

Password = <Director-password> Specifies the password that must be supplied by the above named Director. This directive is required.

Monitor = <yes|no> If Monitor is set to **no** (default), this director will have full access to this Storage daemon. If Monitor is set to **yes**, this director will only be able to fetch the current status of this Storage daemon.

Please note that if this director is being used by a Monitor, we highly recommend to set this directive to **yes** to avoid serious security problems.

The following is an example of a valid Director resource definition:

```
Director {
  Name = MainDirector
  Password = my_secret_password
}
```

7.3 Device Resource

The Device Resource specifies the details of each device (normally a tape drive) that can be used by the Storage daemon. There may be multiple Device resources for a single Storage daemon. In general, the properties specified within the Device resource are specific to the Device.

Name = *Device-Name* Specifies the Name that the Director will use when asking to backup or restore to or from to this device. This is the logical Device name, and may be any string up to 127 characters in length. It is generally a good idea to make it correspond to the English name of the backup device. The physical name of the device is specified on the **Archive Device** directive described below. The name you specify here is also used in your Director's conf file on the Device directive in its Storage resource.

Archive Device = *name-string* The specified **name-string** gives the system file name of the storage device managed by this storage daemon. This will usually be the device file name of a removable storage device (tape drive), for example "**/dev/nst0**" or "**/dev/rmt/0mbn**". For a DVD-writer, it will be for example **/dev/hdc**. It may also be a directory name if you are archiving to disk storage. In this case, you must supply the full absolute path to the directory. When specifying a tape device, it is preferable that the "non-rewind" variant of the device file name be given. In addition, on systems such as Sun, which have multiple tape access methods, you must be sure to specify to use Berkeley

I/O conventions with the device. The **b** in the Solaris (Sun) archive specification **/dev/rmt/0mbn** is what is needed in this case. Bacula does not support SysV tape drive behavior.

As noted above, normally the Archive Device is the name of a tape drive, but you may also specify an absolute path to an existing directory. If the Device is a directory Bacula will write to file storage in the specified directory, and the filename used will be the Volume name as specified in the Catalog. If you want to write into more than one directory (i.e. to spread the load to different disk drives), you will need to define two Device resources, each containing an Archive Device with a different directory. In addition to a tape device name or a directory name, Bacula will accept the name of a FIFO. A FIFO is a special kind of file that connects two programs via kernel memory. If a FIFO device is specified for a backup operation, you must have a program that reads what Bacula writes into the FIFO. When the Storage daemon starts the job, it will wait for **MaximumOpenWait** seconds for the read program to start reading, and then time it out and terminate the job. As a consequence, it is best to start the read program at the beginning of the job perhaps with the **RunBeforeJob** directive. For this kind of device, you never want to specify **AlwaysOpen**, because you want the Storage daemon to open it only when a job starts, so you must explicitly set it to **No**. Since a FIFO is a one way device, Bacula will not attempt to read a label of a FIFO device, but will simply write on it. To create a FIFO Volume in the catalog, use the **add** command rather than the **label** command to avoid attempting to write a label.

```
Device {
  Name = FifoStorage
  Media Type = Fifo
  Device Type = Fifo
  Archive Device = /tmp/fifo
  LabelMedia = yes
  Random Access = no
  AutomaticMount = no
  RemovableMedia = no
  MaximumOpenWait = 60
  AlwaysOpen = no
}
```

During a restore operation, if the Archive Device is a FIFO, Bacula will attempt to read from the FIFO, so you must have an external program that writes into the FIFO. Bacula will wait **MaximumOpenWait** seconds for the program to begin writing and will then time it out and terminate the job. As noted above, you may use the **RunBeforeJob** to start the writer program at the beginning of the job.

The Archive Device directive is required.

Device Type = *type-specification* The Device Type specification allows you to explicitly tell Bacula what kind of device you are defining. It the *type-specification* may be one of the following:

File Tells Bacula that the device is a file. It may either be a file defined on fixed medium or a removable filesystem such as USB. All files must be random access devices.

Tape The device is a tape device and thus is sequential access. Tape devices are controlled using ioctl() calls.

Fifo The device is a first-in-first out sequential access read-only or write-only device.

DVD The device is a DVD. DVDs are sequential access for writing, but random access for reading.

The Device Type directive is not required, and if not specified, Bacula will attempt to guess what kind of device has been specified using the Archive Device specification supplied. There are several advantages to explicitly specifying the Device Type. First, on some systems, block and character devices have the same type, which means that on those systems, Bacula is unlikely to be able to correctly guess that a device is a DVD. Secondly, if you explicitly specify the Device Type, the mount point need not be defined until the device is opened. This is the case with most removable devices such as USB that are mounted by the HAL daemon. If the Device Type is not explicitly specified, then the mount point must exist when the Storage daemon starts.

This directive was implemented in Bacula version 1.38.6.

Media Type = *name-string* The specified **name-string** names the type of media supported by this device, for example, "DLT7000". Media type names are arbitrary in that you set them to anything you want, but they must be known to the volume database to keep track of which storage daemons can read which volumes. In general, each different storage type should have a unique Media Type associated with it. The same **name-string** must appear in the appropriate Storage resource definition in the Director's configuration file.

Even though the names you assign are arbitrary (i.e. you choose the name you want), you should take care in specifying them because the Media Type is used to determine which storage device Bacula will select during restore. Thus you should probably use the same Media Type specification for all drives where the Media can be freely interchanged. This is not generally an issue if you have a single Storage daemon, but it is with multiple Storage daemons, especially if they have incompatible media.

For example, if you specify a Media Type of "DDS-4" then during the restore, Bacula will be able to choose any Storage Daemon that handles "DDS-4". If you have an autochanger, you might want to name the Media Type in a way that is unique to the autochanger, unless you wish to possibly use the Volumes in other drives. You should also ensure to have unique Media Type names if the Media is not compatible between drives. This specification is required for all devices.

In addition, if you are using disk storage, each Device resource will generally have a different mount point or directory. In order for Bacula to select the correct Device resource, each one must have a unique Media Type.

Autochanger = *yes|no* If **Yes**, this device belongs to an automatic tape changer, and you must specify an **Autochanger** resource that points to the **Device** resources. You must also specify a **Changer Device**. If the Autochanger directive is set to **No** (default), the volume must be manually changed. You should also have an identical directive to the Storage resource in the Director's configuration file so that when labeling tapes you are prompted for the slot.

Changer Device = *name-string* The specified **name-string** must be the **generic SCSI** device name of the autochanger that corresponds to the normal read/write **Archive Device** specified in the Device resource. This generic SCSI device name should be specified if you have an autochanger or if you have a standard tape drive and want to use the **Alert Command** (see below). For example, on Linux systems, for an Archive Device name of **/dev/nst0**, you would specify **/dev/sg0** for the Changer Device name. Depending on your exact configuration, and the number of autochangers or the type of autochanger, what you specify here can vary. This directive is optional. See the Using Autochangers chapter of this manual for more details of using this and the following autochanger directives.

Changer Command = *name-string* The **name-string** specifies an external program to be called that will automatically change volumes as required by **Bacula**. Normally, this directive will be specified only in the **AutoChanger** resource, which is then used for all devices. However, you may also specify the different **Changer Command** in each Device resource. Most frequently, you will specify the Bacula supplied **mtx-changer** script as follows:

```
Changer Command = "/path/mtx-changer %c %o %S %a %d"
```

and you will install the **mtx** on your system (found in the **depkgs** release). An example of this command is in the default bacula-sd.conf file. For more details on the substitution characters that may be specified to configure your autochanger please see the Autochangers chapter of this manual. For FreeBSD users, you might want to see one of the several **chio** scripts in **examples/autochangers**.

Alert Command = *name-string* The **name-string** specifies an external program to be called at the completion of each Job after the device is released. The purpose of this command is to check for Tape Alerts, which are present when something is wrong with your tape drive (at least for most modern tape drives). The same substitution characters that may be specified in the Changer Command may also be used in this string. For more information, please see the Autochangers chapter of this manual.

Note, it is not necessary to have an autochanger to use this command. The example below uses the **tapeinfo** program that comes with the **mtx** package, but it can be used on any tape drive. However, you will need to specify a **Changer Device** directive in your Device resource (see above) so that the generic SCSI device name can be edited into the command (with the %c).

An example of the use of this command to print Tape Alerts in the Job report is:

```
Alert Command = "sh -c 'tapeinfo -f %c | grep TapeAlert'"
```

and an example output when there is a problem could be:

```
bacula-sd  Alert: TapeAlert[32]: Interface: Problem with SCSI interface
                  between tape drive and initiator.
```

Drive Index = *number* The **Drive Index** that you specify is passed to the **mtx-changer** script and is thus passed to the **mtx** program. By default, the Drive Index is zero, so if you have only one drive in your autochanger, everything will work normally. However, if you have multiple drives, you must specify multiple Bacula Device resources (one for each drive). The first Device should have the Drive Index set to 0, and the second Device Resource should contain a Drive Index set to 1, and so on. This will then permit you to use two or more drives in your autochanger. As of Bacula version 1.38.0, using the **Autochanger** resource, Bacula will automatically ensure that only one drive at a time uses the autochanger script, so you no longer need locking scripts as in the past – the default mtx-changer script works for any number of drives.

Autoselect = *yes|no* If this directive is set to **yes** (default), and the Device belongs to an autochanger, then when the Autochanger is referenced by the Director, this device can automatically be selected. If this directive is set to **no**, then the Device can only be referenced by directly using the Device name in the Director. This is useful for reserving a drive for something special such as a high priority backup or restore operations.

Maximum Changer Wait = *time* This directive specifies the maximum time in seconds for Bacula to wait for an autochanger to change the volume. If this time is exceeded, Bacula will invalidate the Volume slot number stored in the catalog and try again. If no additional changer volumes exist, Bacula will ask the operator to intervene. The default is 5 minutes.

Maximum Rewind Wait = *time* This directive specifies the maximum time in seconds for Bacula to wait for a rewind before timing out. If this time is exceeded, Bacula will cancel the job. The default is 5 minutes.

Maximum Open Wait = *time* This directive specifies the maximum time in seconds for Bacula to wait for a open before timing out. If this time is exceeded, Bacula will cancel the job. The default is 5 minutes.

Always Open = *yes|no* If **Yes** (default), Bacula will always keep the device open unless specifically **unmounted** by the Console program. This permits Bacula to ensure that the tape drive is always available, and properly positioned. If you set **AlwaysOpen** to **no Bacula** will only open the drive when necessary, and at the end of the Job if no other Jobs are using the drive, it will be freed. The next time Bacula wants to append to a tape on a drive that was freed, Bacula will rewind the tape and position it to the end. To avoid unnecessary tape positioning and to minimize unnecessary operator intervention, it is highly recommended that **Always Open = yes**. This also ensures that the drive is available when Bacula needs it.

If you have **Always Open = yes** (recommended) and you want to use the drive for something else, simply use the **unmount** command in the Console program to release the drive. However, don't forget to remount the drive with **mount** when the drive is available or the next Bacula job will block.

For File storage, this directive is ignored. For a FIFO storage device, you must set this to **No**.

Please note that if you set this directive to **No** Bacula will release the tape drive between each job, and thus the next job will rewind the tape and position it to the end of the data. This can be a very time consuming operation. In addition, with this directive set to no, certain multiple drive autochanger operations will fail. We strongly recommend to keep **Always Open** set to **Yes**

Volume Poll Interval = *time* If the time specified on this directive is non-zero, after asking the operator to mount a new volume Bacula will periodically poll (or read) the drive at the specified interval to see if a new volume has been mounted. If the time interval is zero (the default), no polling will occur. This directive can be useful if you want to avoid operator intervention via the console. Instead, the operator can simply remove the old volume and insert the requested one, and Bacula on the next poll will recognize the new tape and continue. Please be aware that if you set this interval too small, you may excessively wear your tape drive if the old tape remains in the drive, since Bacula will read it on each poll. This can be avoided by ejecting the tape using the **Offline On Unmount** and the **Close on Poll** directives. However, if you are using a Linux 2.6 kernel or other OSes such as FreeBSD or Solaris, the Offline On Unmount will leave the drive with no tape, and Bacula will not be able to properly open the drive and may fail the job. For more information on this problem, please see the description of Offline On Unmount in the Tape Testing chapter.

Close on Poll= *yes|no* If **Yes**, Bacula close the device (equivalent to an unmount except no mount is required) and reopen it at each poll. Normally this is not too useful unless you have the **Offline on Unmount** directive set, in which case the drive will be taken offline preventing wear on the tape during any future polling. Once the operator inserts a new tape, Bacula will recognize the drive on the next poll and automatically continue with the backup. Please see above more more details.

Maximum Open Wait = *time* This directive specifies the maximum amount of time in seconds that Bacula will wait for a device that is busy. The default is 5 minutes. If the device cannot be obtained, the current Job will be terminated in error. Bacula will re-attempt to open the drive the next time a Job starts that needs the the drive.

Removable media = *yes|no* If **Yes**, this device supports removable media (for example, tapes or CDs). If **No**, media cannot be removed (for example, an intermediate backup area on a hard disk). If **Removable media** is enabled on a File device (as opposed to a tape) the Storage daemon will assume that device may be something like a USB device that can be removed or a simply a removable harddisk. When attempting to open such a device, if the Volume is not found (for File devices, the Volume name is the same as the Filename), then the Storage daemon will search the entire device looking for likely Volume names, and for each one found, it will ask the Director if the Volume can be used. If so, the Storage daemon will use the first such Volume found. Thus it acts somewhat like a tape drive – if the correct Volume is not found, it looks at what actually is found, and if it is an appendable Volume, it will use it.

If the removable medium is not automatically mounted (e.g. udev), then you might consider using additional Storage daemon device directives such as **Requires Mount**, **Mount Point**, **Mount Command**, and **Unmount Command**, all of which can be used in conjunction with **Removable Media**.

Random access = *yes|no* If **Yes**, the archive device is assumed to be a random access medium which supports the **lseek** (or **lseek64** if Largefile is enabled during configuration) facility. This should be set to **Yes** for all file systems such as DVD, USB, and fixed files. It should be set to **No** for non-random access devices such as tapes and named pipes.

Requires Mount = *yes|no* When this directive is enabled, the Storage daemon will submit a **Mount Command** before attempting to open the device. You must set this directive to **yes** for DVD-writers and removable file systems such as USB devices that are not automatically mounted by the operating system when plugged in or opened by Bacula. It should be set to **no** for all other devices such as tapes and fixed filesystems. It should also be set to **no** for any removable device that is automatically mounted by the operating system when opened (e.g. USB devices mounted by udev or hotplug). This directive indicates if the device requires to be mounted using the **Mount Command**. To be able to write a DVD, the following directives must also be defined: **Mount Point**, **Mount Command**, **Unmount Command** and **Write Part Command**.

Mount Point = *directory* Directory where the device can be mounted. This directive is used only for devices that have **Requires Mount** enabled such as DVD or USB file devices.

Mount Command = *name-string* This directive specifies the command that must be executed to mount devices such as DVDs and many USB devices. For DVDs, the device is written directly, but the mount command is necessary in order to determine the free space left on the DVD. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, for a DVD, you will define it as follows:

```
Mount Command = "/bin/mount -t iso9660 -o ro %a %m"
```

However, if you have defined a mount point in /etc/fstab, you might be able to use a mount command such as:

```
Mount Command = "/bin/mount /media/dvd"
```

See the Edit Codes section below for more details of the editing codes that can be used in this directive.

If you need to specify multiple commands, create a shell script.

Unmount Command = *name-string* This directive specifies the command that must be executed to unmount devices such as DVDs and many USB devices. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Unmount Command = "/bin/umount %m"
```

See the Edit Codes section below for more details of the editing codes that can be used in this directive.

If you need to specify multiple commands, create a shell script.

Minimum block size = *size-in-bytes* On most modern tape drives, you will not need or want to specify this directive, and if you do so, it will be to make Bacula use fixed block sizes. This statement applies only to non-random access devices (e.g. tape drives). Blocks written by the storage daemon to a non-random archive device will never be smaller than the given **size-in-bytes**. The Storage daemon will attempt to efficiently fill blocks with data received from active sessions but will, if necessary, add padding to a block to achieve the required minimum size.

To force the block size to be fixed, as is the case for some non-random access devices (tape drives), set the **Minimum block size** and the **Maximum block size** to the same value (zero included). The default is that both the minimum and maximum block size are zero and the default block size is 64,512 bytes.

For example, suppose you want a fixed block size of 100K bytes, then you would specify:

```
Minimum block size = 100K
Maximum block size = 100K
```

Please note that if you specify a fixed block size as shown above, the tape drive must either be in variable block size mode, or if it is in fixed block size mode, the block size (generally defined by **mt**) **must** be identical to the size specified in Bacula – otherwise when you attempt to re-read your Volumes, you will get an error.

If you want the block size to be variable but with a 64K minimum and 200K maximum (and default as well), you would specify:

```
Minimum block size = 64K
Maximum blocksize = 200K
```

Maximum block size = *size-in-bytes* On most modern tape drives, you will not need to specify this directive. If you do so, it will most likely be to use fixed block sizes (see Minimum block size above). The Storage daemon will always attempt to write blocks of the specified **size-in-bytes** to the archive device. As a consequence, this statement specifies both the default block size and the maximum block size. The size written never exceed the given **size-in-bytes**. If adding data to a block would cause it to exceed the given maximum size, the block will be written to the archive device, and the new data will begin a new block.

If no value is specified or zero is specified, the Storage daemon will use a default block size of 64,512 bytes (126 * 512).

The maximum **size-in-bytes** possible is 2,000,000.

Hardware End of Medium = *yes|no* If **No**, the archive device is not required to support end of medium ioctl request, and the storage daemon will use the forward space file function to find the end of the recorded data. If **Yes**, the archive device must support the `ioctl MTEOM` call, which will position the tape to the end of the recorded data. In addition, your SCSI driver must keep track of the file number on the tape and report it back correctly by the **MTIOCGET** ioctl. Note, some SCSI drivers will correctly forward space to the end of the recorded data, but they do not keep track of the file number. On Linux machines, the SCSI driver has a **fast-eod** option, which if set will cause the driver to lose track of the file number. You should ensure that this option is always turned off using the **mt** program.

Default setting for Hardware End of Medium is **Yes**. This function is used before appending to a tape to ensure that no previously written data is lost. We recommend if you have a non-standard or unusual tape drive that you use the **btape** program to test your drive to see whether or not it supports this function. All modern (after 1998) tape drives support this feature.

Fast Forward Space File = *yes|no* If **No**, the archive device is not required to support keeping track of the file number (**MTIOCGET** ioctl) during forward space file. If **Yes**, the archive device must support the `ioctl MTFSF` call, which virtually all drivers support, but in addition, your SCSI driver must keep track of the file number on the tape and report it back correctly by the **MTIOCGET** ioctl. Note, some SCSI drivers will correctly forward space, but they do not keep track of the file number or more seriously, they do not report end of medium.

Default setting for Fast Forward Space File is **Yes**.

Use MTIOCGET = *yes|no* If **No**, the operating system is not required to support keeping track of the file number and reporting it in the (**MTIOCGET** ioctl). The default is **Yes**. If you must set this to No, Bacula will do the proper file position determination, but it is very unfortunate because it means that tape movement is very inefficient. Fortunately, this operation system deficiency seems to be the case only on a few *BSD systems. Operating systems known to work correctly are Solaris, Linux and FreeBSD.

BSF at EOM = *yes|no* If **No**, the default, no special action is taken by Bacula with the End of Medium (end of tape) is reached because the tape will be positioned after the last EOF tape mark, and Bacula can append to the tape as desired. However, on some systems, such as FreeBSD, when Bacula reads the End of Medium (end of tape), the tape will be positioned after the second EOF tape mark (two successive EOF marks indicated End of Medium). If Bacula appends from that point, all the appended data will be lost. The solution for such systems is to specify **BSF at EOM** which causes Bacula to backspace over the second EOF mark. Determination of whether or not you need this directive is done using the **test** command in the **btape** program.

TWO EOF = *yes|no* If **Yes**, Bacula will write two end of file marks when terminating a tape – i.e. after the last job or at the end of the medium. If **No**, the default, Bacula will only write one end of file to terminate the tape.

Backward Space Record = *yes|no* If *Yes*, the archive device supports the `MTBSR ioctl` to backspace records. If *No*, this call is not used and the device must be rewound and advanced forward to the desired position. Default is **Yes** for non random-access devices. This function if enabled is used at the end of a Volume after writing the end of file and any ANSI/IBM labels to determine whether or not the last block was written correctly. If you turn this function off, the test will not be done. This causes no harm as the re-read process is precautionary rather than required.

Backward Space File = *yes|no* If *Yes*, the archive device supports the **MTBSF** and **MTBSF ioctl**s to backspace over an end of file mark and to the start of a file. If *No*, these calls are not used and the device must be rewound and advanced forward to the desired position. Default is **Yes** for non random-access devices.

Forward Space Record = *yes|no* If *Yes*, the archive device must support the **MTFSR ioctl** to forward space over records. If **No**, data must be read in order to advance the position on the device. Default is **Yes** for non random-access devices.

Forward Space File = *yes|no* If **Yes**, the archive device must support the `MTFSF ioctl` to forward space by file marks. If *No*, data must be read to advance the position on the device. Default is **Yes** for non random-access devices.

Offline On Unmount = *yes|no* The default for this directive is **No**. If **Yes** the archive device must support the `MTOFFL ioctl` to rewind and take the volume offline. In this case, Bacula will issue the offline (eject) request before closing the device during the **unmount** command. If **No** Bacula will not attempt to offline the device before unmounting it. After an offline is issued, the cassette will be ejected thus **requiring operator intervention** to continue, and on some systems require an explicit load command to be issued (**mt -f /dev/xxx load**) before the system will recognize the tape. If you are using an autochanger, some devices require an offline to be issued prior to changing the volume. However, most devices do not and may get very confused.

If you are using a Linux 2.6 kernel or other OSes such as FreeBSD or Solaris, the Offline On Unmount will leave the drive with no tape, and Bacula will not be able to properly open the drive and may fail the job. For more information on this problem, please see the description of Offline On Unmount in the Tape Testing chapter.

Maximum Concurrent Jobs = <number> where <number> is the maximum number of Jobs that can run concurrently on a specified Device. Using this directive, it is possible to have different Jobs using multiple drives, because when the Maximum Concurrent Jobs limit is reached, the Storage Daemon will start new Jobs on any other available compatible drive. This facilitates writing to multiple drives with multiple Jobs that all use the same Pool.

Maximum Volume Size = *size* No more than **size** bytes will be written onto a given volume on the archive device. This directive is used mainly in testing Bacula to simulate a small Volume. It can also be useful if you wish to limit the size of a File Volume to say less than 2GB of data. In some rare cases of really antiquated tape drives that do not properly indicate when the end of a tape is reached during writing (though I have read about such drives, I have never personally encountered one). Please note, this directive is deprecated (being phased out) in favor of the **Maximum Volume Bytes** defined in the Director's configuration file.

Maximum File Size = *size* No more than **size** bytes will be written into a given logical file on the volume. Once this size is reached, an end of file mark is written on the volume and subsequent data are written into the next file. Breaking long sequences of data blocks with file marks permits quicker positioning to the start of a given stream of data and can improve recovery from read errors on the volume. The default is one Gigabyte. This directive creates EOF marks only on tape media. However, regardless of the medium type (tape, disk, DVD, ...) each time a the Maximum File Size is exceeded, a record is put into the catalog database that permits seeking to that position on the medium for restore operations. If you set this to a small value (e.g. 1MB), you will generate lots of database records (JobMedia) and may significantly increase CPU/disk overhead.

If you are configuring an LTO-3 or LTO-4 tape, you probably will want to set the **Maximum File Size** to 2GB to avoid making the drive stop to write an EOF mark.

Note, this directive does not limit the size of Volumes that Bacula will create regardless of whether they are tape or disk volumes. It changes only the number of EOF marks on a tape and the number of block positioning records (see below) that are generated. If you want to limit the size of all Volumes for a particular device, use the **Maximum Volume Size** directive (above), or use the **Maximum Volume Bytes** directive in the Director's Pool resource, which does the same thing but on a Pool (Volume) basis.

Block Positioning = *yes|no* This directive tells Bacula not to use block positioning when doing restores. Turning this directive off can cause Bacula to be **extremely** slow when restoring files. You might use this directive if you wrote your tapes with Bacula in variable block mode (the default), but your drive was in fixed block mode. The default is **yes**.

Maximum Network Buffer Size = *bytes* where *bytes* specifies the initial network buffer size to use with the File daemon. This size will be adjusted down if it is too large until it is accepted by the OS. Please use care in setting this value since if it is too large, it will be trimmed by 512 bytes until the OS is happy, which may require a large number of system calls. The default value is 32,768 bytes.

The default size was chosen to be relatively large but not too big in the case that you are transmitting data over Internet. It is clear that on a high speed local network, you can increase this number and improve performance. For example, some users have found that if you use a value of 65,536 bytes they get five to ten times the throughput. Larger values for most users don't seem to improve performance. If you are interested in improving your backup speeds, this is definitely a place to experiment. You will probably also want to make the corresponding change in each of your File daemons conf files.

Maximum Spool Size = *bytes* where the bytes specify the maximum spool size for all jobs that are running. The default is no limit.

Maximum Job Spool Size = *bytes* where the bytes specify the maximum spool size for any one job that is running. The default is no limit. This directive is implemented only in version 1.37 and later.

Spool Directory = *directory* specifies the name of the directory to be used to store the spool files for this device. This directory is also used to store temporary part files when writing to a device that requires mount (DVD). The default is to use the working directory.

Maximum Part Size = *bytes* This is the maximum size of a volume part file. The default is no limit. This directive is implemented only in version 1.37 and later.

If the device requires mount, it is transferred to the device when this size is reached. In this case, you must take care to have enough disk space left in the spool directory.

Otherwise, it is left on the hard disk.

It is ignored for tape and FIFO devices.

7.4 Edit Codes for Mount and Unmount Directives

Before submitting the **Mount Command**, **Unmount Command**, **Write Part Command**, or **Free Space Command** directives to the operating system, Bacula performs character substitution of the following characters:

```
%% = %
%a = Archive device name
%e = erase (set if cannot mount and first part)
%n = part number
%m = mount point
%v = last part name (i.e. filename)
```

7.5 Devices that require a mount (DVD)

All the directives in this section are implemented only in Bacula version 1.37 and later and hence are available in version 1.38.6.

As of version 1.39.5, the directives "Requires Mount", "Mount Point", "Mount Command", and "Unmount Command" apply to removable filesystems such as USB in addition to DVD.

Requires Mount = *yes|no* You must set this directive to **yes** for DVD-writers, and to **no** for all other devices (tapes/files). This directive indicates if the device requires to be mounted to be read, and if it must be written in a special way. If it set, **Mount Point**, **Mount Command**, **Unmount Command** and **Write Part Command** directives must also be defined.

Mount Point = *directory* Directory where the device can be mounted.

Mount Command = *name-string* Command that must be executed to mount the device. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Mount Command = "/bin/mount -t iso9660 -o ro %a %m"
```

For some media, you may need multiple commands. If so, it is recommended that you use a shell script instead of putting them all into the Mount Command. For example, instead of this:

```
Mount Command = "/usr/local/bin/mymount"
```

Where that script contains:

```
#!/bin/sh
ndasadmin enable -s 1 -o w
sleep 2
mount /dev/ndas-00323794-0p1 /backup
```

Similar consideration should be given to all other Command parameters.

Unmount Command = *name-string* Command that must be executed to unmount the device. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Unmount Command = "/bin/umount %m"
```

If you need to specify multiple commands, create a shell script.

Write Part Command = *name-string* Command that must be executed to write a part to the device. Before the command is executed, %a is replaced with the Archive Device, %m with the Mount Point, %e is replaced with 1 if we are writing the first part, and with 0 otherwise, and %v with the current part filename.

For a DVD, you will most frequently specify the Bacula supplied **dvd-handler** script as follows:

```
Write Part Command = "/path/dvd-handler %a write %e %v"
```

Where **/path** is the path to your scripts install directory, and dvd-handler is the Bacula supplied script file. This command will already be present, but commented out, in the default bacula-sd.conf file. To use it, simply remove the comment (#) symbol.

If you need to specify multiple commands, create a shell script.

Free Space Command = *name-string* Command that must be executed to check how much free space is left on the device. Before the command is executed,%a is replaced with the Archive Device, %m with the Mount Point, %e is replaced with 1 if we are writing the first part, and with 0 otherwise, and %v with the current part filename.

For a DVD, you will most frequently specify the Bacula supplied **dvd-handler** script as follows:

```
Free Space Command = "/path/dvd-handler %a free"
```

Where **/path** is the path to your scripts install directory, and dvd-handler is the Bacula supplied script file. If you want to specify your own command, please look at the code of dvd-handler to see what output Bacula expects from this command. This command will already be present, but commented out, in the default bacula-sd.conf file. To use it, simply remove the comment (#) symbol.

If you do not set it, Bacula will expect there is always free space on the device.

If you need to specify multiple commands, create a shell script.

Chapter 8

Autochanger Resource

The Autochanger resource supports single or multiple drive autochangers by grouping one or more Device resources into one unit called an autochanger in Bacula (often referred to as a "tape library" by autochanger manufacturers).

If you have an Autochanger, and you want it to function correctly, you **must** have an Autochanger resource in your Storage conf file, and your Director's Storage directives that want to use an Autochanger **must** refer to the Autochanger resource name. In previous versions of Bacula, the Director's Storage directives referred directly to Device resources that were autochangers. In version 1.38.0 and later, referring directly to Device resources will not work for Autochangers.

Name = <Autochanger-Name> Specifies the Name of the Autochanger. This name is used in the Director's Storage definition to refer to the autochanger. This directive is required.

Device = <Device-name1, device-name2, ...> Specifies the names of the Device resource or resources that correspond to the autochanger drive. If you have a multiple drive autochanger, you must specify multiple Device names, each one referring to a separate Device resource that contains a Drive Index specification that corresponds to the drive number base zero. You may specify multiple device names on a single line separated by commas, and/or you may specify multiple Device directives. This directive is required.

Changer Device = *name-string* The specified **name-string** gives the system file name of the autochanger device name. If specified in this resource, the Changer Device name is not needed in the Device resource. If it is specified in the Device resource (see above), it will take precedence over one specified in the Autochanger resource.

Changer Command = *name-string* The **name-string** specifies an external program to be called that will automatically change volumes as required by **Bacula**. Most frequently, you will specify the Bacula supplied **mtx-changer** script as follows. If it is specified here, it need not be specified in the Device resource. If it is also specified in the Device resource, it will take precedence over the one specified in the Autochanger resource.

The following is an example of a valid Autochanger resource definition:

```
Autochanger {
  Name = "DDS-4-changer"
  Device = DDS-4-1, DDS-4-2, DDS-4-3
  Changer Device = /dev/sg0
  Changer Command = "/etc/bacula/mtx-changer %c %o %S %a %d"
}
Device {
  Name = "DDS-4-1"
  Drive Index = 0
  Autochanger = yes
  ...
}
```

```
Device {
  Name = "DDS-4-2"
  Drive Index = 1
  Autochanger = yes
  ...
Device {
  Name = "DDS-4-3"
  Drive Index = 2
  Autochanger = yes
  Autoselect = no
  ...
}
```

Please note that it is important to include the **Autochanger = yes** directive in each Device definition that belongs to an Autochanger. A device definition should not belong to more than one Autochanger resource. Also, your Device directive in the Storage resource of the Director's conf file should have the Autochanger's resource name rather than a name of one of the Devices.

If you have a drive that physically belongs to an Autochanger but you don't want to have it automatically used when Bacula references the Autochanger for backups, for example, you want to reserve it for restores, you can add the directive:

```
Autoselect = no
```

to the Device resource for that drive. In that case, Bacula will not automatically select that drive when accessing the Autochanger. You can, still use the drive by referencing it by the Device name directly rather than the Autochanger name. An example of such a definition is shown above for the Device DDS-4-3, which will not be selected when the name DDS-4-changer is used in a Storage definition, but will be used if DDS-4-3 is used.

8.1 Capabilities

Label media = *yes|no* If **Yes**, permits this device to automatically label blank media without an explicit operator command. It does so by using an internal algorithm as defined on the Label Format record in each Pool resource. If this is **No** as by default, Bacula will label tapes only by specific operator command (**label** in the Console) or when the tape has been recycled. The automatic labeling feature is most useful when writing to disk rather than tape volumes.

Automatic mount = *yes|no* If **Yes** (the default), permits the daemon to examine the device to determine if it contains a Bacula labeled volume. This is done initially when the daemon is started, and then at the beginning of each job. This directive is particularly important if you have set **Always Open = no** because it permits Bacula to attempt to read the device before asking the system operator to mount a tape. However, please note that the tape must be mounted before the job begins.

8.2 Messages Resource

For a description of the Messages Resource, please see the Messages Resource Chapter of this manual.

8.3 Sample Storage Daemon Configuration File

A example Storage Daemon configuration file might be the following:

```
#
# Default Bacula Storage Daemon Configuration file
```

```
#
#  For Bacula release 1.37.2 (07 July 2005) -- gentoo 1.4.16
#
# You may need to change the name of your tape drive
#   on the "Archive Device" directive in the Device
#   resource.  If you change the Name and/or the
#   "Media Type" in the Device resource, please ensure
#   that bacula-dir.conf has corresponding changes.
#
Storage {                               # definition of myself
  Name = rufus-sd
  Address = rufus
  WorkingDirectory = "$HOME/bacula/bin/working"
  Pid Directory = "$HOME/bacula/bin/working"
  Maximum Concurrent Jobs = 20
}
#
# List Directors who are permitted to contact Storage daemon
#
Director {
  Name = rufus-dir
  Password = "ZF9Ctf5PQoWCPkmR3s4atCB0usUPg+vWWyIo2VS5ti6k"
}
#
# Restricted Director, used by tray-monitor to get the
#   status of the storage daemon
#
Director {
  Name = rufus-mon
  Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
  Monitor = yes
}
#
# Devices supported by this Storage daemon
# To connect, the Director's bacula-dir.conf must have the
#  same Name and MediaType.
#
Autochanger {
  Name = Autochanger
  Device = Drive-1
  Device = Drive-2
  Changer Command = "/home/kern/bacula/bin/mtx-changer %c %o %S %a %d"
  Changer Device = /dev/sg0
}

Device {
  Name = Drive-1                      #
  Drive Index = 0
  Media Type = DLT-8000
  Archive Device = /dev/nst0
  AutomaticMount = yes;               # when device opened, read it
  AlwaysOpen = yes;
  RemovableMedia = yes;
  RandomAccess = no;
  AutoChanger = yes
  Alert Command = "sh -c 'tapeinfo -f %c |grep TapeAlert|cat'"
}

Device {
  Name = Drive-2                      #
  Drive Index = 1
  Media Type = DLT-8000
  Archive Device = /dev/nst1
  AutomaticMount = yes;               # when device opened, read it
  AlwaysOpen = yes;
  RemovableMedia = yes;
  RandomAccess = no;
  AutoChanger = yes
  Alert Command = "sh -c 'tapeinfo -f %c |grep TapeAlert|cat'"
}

Device {
  Name = "HP DLT 80"
  Media Type = DLT8000
  Archive Device = /dev/nst0
  AutomaticMount = yes;                 # when device opened, read it
```

```
  AlwaysOpen = yes;
  RemovableMedia = yes;
}
#Device {
#  Name = SDT-7000                     #
#  Media Type = DDS-2
#  Archive Device = /dev/nst0
#  AutomaticMount = yes;               # when device opened, read it
#  AlwaysOpen = yes;
#  RemovableMedia = yes;
#}
#Device {
#  Name = Floppy
#  Media Type = Floppy
#  Archive Device = /mnt/floppy
#  RemovableMedia = yes;
#  Random Access = Yes;
#  AutomaticMount = yes;               # when device opened, read it
#  AlwaysOpen = no;
#}
#Device {
#  Name = FileStorage
#  Media Type = File
#  Archive Device = /tmp
#  LabelMedia = yes;                   # lets Bacula label unlabeled media
#  Random Access = Yes;
#  AutomaticMount = yes;               # when device opened, read it
#  RemovableMedia = no;
#  AlwaysOpen = no;
#}
#Device {
#  Name = "NEC ND-1300A"
#  Media Type = DVD
#  Archive Device = /dev/hda
#  LabelMedia = yes;                   # lets Bacula label unlabeled media
#  Random Access = Yes;
#  AutomaticMount = yes;               # when device opened, read it
#  RemovableMedia = yes;
#  AlwaysOpen = no;
#  MaximumPartSize = 800M;
#  RequiresMount = yes;
#  MountPoint = /mnt/cdrom;
#  MountCommand = "/bin/mount -t iso9660 -o ro %a %m";
#  UnmountCommand = "/bin/umount %m";
#  SpoolDirectory = /tmp/backup;
#  WritePartCommand = "/etc/bacula/dvd-handler %a write %e %v"
#  FreeSpaceCommand = "/etc/bacula/dvd-handler %a free"
#}
#
# A very old Exabyte with no end of media detection
#
#Device {
#  Name = "Exabyte 8mm"
#  Media Type = "8mm"
#  Archive Device = /dev/nst0
#  Hardware end of medium = No;
#  AutomaticMount = yes;               # when device opened, read it
#  AlwaysOpen = Yes;
#  RemovableMedia = yes;
#}
#
# Send all messages to the Director,
# mount messages also are sent to the email address
#
Messages {
  Name = Standard
  director = rufus-dir = all
  operator = root = mount
}
```

Chapter 9

Messages Resource

The Messages resource defines how messages are to be handled and destinations to which they should be sent.

Even though each daemon has a full message handler, within the File daemon and the Storage daemon, you will normally choose to send all the appropriate messages back to the Director. This permits all the messages associated with a single Job to be combined in the Director and sent as a single email message to the user, or logged together in a single file.

Each message that Bacula generates (i.e. that each daemon generates) has an associated type such as INFO, WARNING, ERROR, FATAL, etc. Using the message resource, you can specify which message types you wish to see and where they should be sent. In addition, a message may be sent to multiple destinations. For example, you may want all error messages both logged as well as sent to you in an email. By defining multiple messages resources, you can have different message handling for each type of Job (e.g. Full backups versus Incremental backups).

In general, messages are attached to a Job and are included in the Job report. There are some rare cases, where this is not possible, e.g. when no job is running, or if a communications error occurs between a daemon and the director. In those cases, the message may remain in the system, and should be flushed at the end of the next Job. However, since such messages are not attached to a Job, any that are mailed will be sent to **/usr/lib/sendmail**. On some systems, such as FreeBSD, if your sendmail is in a different place, you may want to link it to the the above location.

The records contained in a Messages resource consist of a **destination** specification followed by a list of **message-types** in the format:

destination = message-type1, message-type2, message-type3, ...

or for those destinations that need and address specification (e.g. email):

destination = address = message-type1, message-type2, message-type3, ... Where **destination** is one of a predefined set of keywords that define where the message is to be sent (**stdout**, **file**, ...), **message-type** is one of a predefined set of keywords that define the type of message generated by **Bacula** (**ERROR**, **WARNING**, **FATAL**, ...), and **address** varies according to the **destination** keyword, but is typically an email address or a filename.

The following are the list of the possible record definitions that can be used in a message resource.

Messages Start of the Messages records.

Name = <name> The name of the Messages resource. The name you specify here will be used to tie this Messages resource to a Job and/or to the daemon.

MailCommand = <command> In the absence of this resource, Bacula will send all mail using the following command:

mail -s "Bacula Message" <recipients>

In many cases, depending on your machine, this command may not work. Using the **MailCommand**, you can specify exactly how to send the mail. During the processing of the **command** part, normally specified as a quoted string, the following substitutions will be used:

- %% = %
- %c = Client's name
- %d = Director's name
- %e = Job Exit code (OK, Error, ...)
- %i = Job Id
- %j = Unique Job name
- %l = Job level
- %n = Job name
- %r = Recipients
- %t = Job type (e.g. Backup, ...)

Please note: any **MailCommand** directive must be specified in the **Messages** resource **before** the desired **Mail**, **MailOnSuccess**, or **MailOnError** directive. In fact, each of those directives may be preceded by a different **MailCommand**.

The following is the command I (Kern) use. Note, the whole command should appear on a single line in the configuration file rather than split as is done here for presentation:

mailcommand = "/home/kern/bacula/bin/bsmtp -h mail.example.com -f \"\(Bacula\) %r\" -s \"Bacula: %t %e of %c %l\" %r"

The **bsmtp** program is provided as part of **Bacula**. For additional details, please see the bsmtp – Customizing Your Email Messages section of the Bacula Utility Programs chapter of this manual. Please test any **mailcommand** that you use to ensure that your bsmtp gateway accepts the addressing form that you use. Certain programs such as Exim can be very selective as to what forms are permitted particularly in the from part.

OperatorCommand = <command> This resource specification is similar to the **MailCommand** except that it is used for Operator messages. The substitutions performed for the **MailCommand** are also done for this command. Normally, you will set this command to the same value as specified for the **MailCommand**. The **OperatorCommand** directive must appear in the **Messages** resource before the **Operator** directive.

<destination> = <message-type1>, <message-type2>, ... Where **destination** may be one of the following:

stdout Send the message to standard output.

stderr Send the message to standard error.

console Send the message to the console (Bacula Console). These messages are held until the console program connects to the Director.

<destination> = <address> = <message-type1>, <message-type2>, ...

Where **address** depends on the **destination**.

The **destination** may be one of the following:

director Send the message to the Director whose name is given in the **address** field. Note, in the current implementation, the Director Name is ignored, and the message is sent to the Director that started the Job.

file Send the message to the filename given in the **address** field. If the file already exists, it will be overwritten.

append Append the message to the filename given in the **address** field. If the file already exists, it will be appended to. If the file does not exist, it will be created.

syslog Send the message to the system log (syslog) using the facility specified in the **address** field. Note, for the moment, the **address** field is ignored and the message is always sent to the LOG_DAEMON facility with level LOG_ERR. See **man 3 syslog** for more details. Example:

```
syslog = all, !skipped
```

mail Send the message to the email addresses that are given as a comma separated list in the **address** field. Mail messages are grouped together during a job and then sent as a single email message when the job terminates. The advantage of this destination is that you are notified about every Job that runs. However, if you backup five or ten machines every night, the volume of email messages can be important. Some users use filter programs such as **procmail** to automatically file this email based on the Job termination code (see **mailcommand**).

mail on error Send the message to the email addresses that are given as a comma separated list in the **address** field if the Job terminates with an error condition. MailOnError messages are grouped together during a job and then sent as a single email message when the job terminates. This destination differs from the **mail** destination in that if the Job terminates normally, the message is totally discarded (for this destination). If the Job terminates in error, it is emailed. By using other destinations such as **append** you can ensure that even if the Job terminates normally, the output information is saved.

mail on success Send the message to the email addresses that are given as a comma separated list in the **address** field if the Job terminates normally (no error condition). MailOnSuccess messages are grouped together during a job and then sent as a single email message when the job terminates. This destination differs from the **mail** destination in that if the Job terminates abnormally, the message is totally discarded (for this destination). If the Job terminates normally, it is emailed.

operator Send the message to the email addresses that are specified as a comma separated list in the **address** field. This is similar to **mail** above, except that each message is sent as received. Thus there is one email per message. This is most useful for **mount** messages (see below).

console Send the message to the Bacula console.

stdout Send the message to the standard output (normally not used).

stderr Send the message to the standard error output (normally not used).

catalog Send the message to the Catalog database. The message will be written to the table named **Log** and a timestamp field will also be added. This permits Job Reports and other messages to be recorded in the Catalog so that they can be accessed by reporting software. Bacula will prune the Log records associated with a Job when the Job records are pruned. Otherwise, Bacula never uses these records internally, so this destination is only used for special purpose programs (e.g. **bweb**).

For any destination, the **message-type** field is a comma separated list of the following types or classes of messages:

info General information messages.

warning Warning messages. Generally this is some unusual condition but not expected to be serious.

error Non-fatal error messages. The job continues running. Any error message should be investigated as it means that something went wrong.

fatal Fatal error messages. Fatal errors cause the job to terminate.

terminate Message generated when the daemon shuts down.

notsaved Files not saved because of some error. Usually because the file cannot be accessed (i.e. it does not exist or is not mounted).

skipped Files that were skipped because of a user supplied option such as an incremental backup or a file that matches an exclusion pattern. This is not considered an error condition such as the files listed for the **notsaved** type because the configuration file explicitly requests these types of files to be skipped. For example, any unchanged file during an incremental backup, or any subdirectory if the no recursion option is specified.

mount Volume mount or intervention requests from the Storage daemon. These requests require a specific operator intervention for the job to continue.

restored The **ls** style listing generated for each file restored is sent to this message class.

all All message types.

security Security info/warning messages principally from unauthorized connection attempts.

alert Alert messages. These are messages generated by tape alerts.

volmgmt Volume management messages. Currently there are no volume mangement messages generated.

The following is an example of a valid Messages resource definition, where all messages except files explicitly skipped or daemon termination messages are sent by email to enforcement@sec.com. In addition all mount messages are sent to the operator (i.e. emailed to enforcement@sec.com). Finally all messages other than explicitly skipped files and files saved are sent to the console:

```
Messages {
  Name = Standard
  mail = enforcement@sec.com = all, !skipped, !terminate
  operator = enforcement@sec.com = mount
  console = all, !skipped, !saved
}
```

With the exception of the email address (changed to avoid junk mail from robot's), an example Director's Messages resource is as follows. Note, the **mailcommand** and **operatorcommand** are on a single line – they had to be split for this manual:

```
Messages {
  Name = Standard
  mailcommand = "bacula/bin/bsmtp -h mail.example.com \
    -f \"\(Bacula\) %r\" -s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "bacula/bin/bsmtp -h mail.example.com \
    -f \"\(Bacula\) %r\" -s \"Bacula: Intervention needed \
        for %j\" %r"
  MailOnError = security@example.com = all, !skipped, \
                !terminate
  append = "bacula/bin/log" = all, !skipped, !terminate
  operator = security@example.com = mount
  console = all, !skipped, !saved
}
```

Chapter 10

Console Configuration

10.1 General

The Console configuration file is the simplest of all the configuration files, and in general, you should not need to change it except for the password. It simply contains the information necessary to contact the Director or Directors.

For a general discussion of the syntax of configuration files and their resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual.

The following Console Resource definition must be defined:

10.2 The Director Resource

The Director resource defines the attributes of the Director running on the network. You may have multiple Director resource specifications in a single Console configuration file. If you have more than one, you will be prompted to choose one when you start the **Console** program.

Director Start of the Director directives.

Name = <name> The director name used to select among different Directors, otherwise, this name is not used.

DIRPort = <port-number> Specify the port to use to connect to the Director. This value will most likely already be set to the value you specified on the **--with-base-port** option of the **./configure** command. This port must be identical to the **DIRport** specified in the **Director** resource of the Director's configuration file. The default is 9101 so this directive is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director.

Password = <password> Where the password is the password needed for the Director to accept the Console connection. This password must be identical to the **Password** specified in the **Director** resource of the Director's configuration file. This directive is required.

An actual example might be:

```
Director {
  Name = HeadMan
  address = rufus.cats.com
  password = xyz1erploit
}
```

10.3 The ConsoleFont Resource

The ConsoleFont resource is available only in the GNOME version of the console. It permits you to define the font that you want used to display in the main listing window.

ConsoleFont Start of the ConsoleFont directives.

Name = <name> The name of the font.

Font = <Pango Font Name> The string value given here defines the desired font. It is specified in the Pango format. For example, the default specification is:

```
Font = "LucidaTypewriter 9"
```

Thanks to Phil Stracchino for providing the code for this feature.

An different example might be:

```
ConsoleFont {
  Name = Default
  Font = "Monospace 10"
}
```

10.4 The Console Resource

As of Bacula version 1.33 and higher, there are three different kinds of consoles, which the administrator or user can use to interact with the Director. These three kinds of consoles comprise three different security levels.

- The first console type is an **anonymous** or **default** console, which has full privileges. There is no console resource necessary for this type since the password is specified in the Director resource. This is the kind of console that was initially implemented in versions prior to 1.33 and remains valid. Typically you would use it only for administrators.
- The second type of console, and new to version 1.33 and higher is a "named" or "restricted" console defined within a Console resource in both the Director's configuration file and in the Console's configuration file. Both the names and the passwords in these two entries must match much as is the case for Client programs.

  This second type of console begins with absolutely no privileges except those explicitly specified in the Director's Console resource. Note, the definition of what these restricted consoles can do is determined by the Director's conf file.

  Thus you may define within the Director's conf file multiple Consoles with different names and passwords, sort of like multiple users, each with different privileges. As a default, these consoles can do absolutely nothing – no commands what so ever. You give them privileges or rather access to commands and resources by specifying access control lists in the Director's Console resource. This gives the administrator fine grained control over what particular consoles (or users) can do.
- The third type of console is similar to the above mentioned restricted console in that it requires a Console resource definition in both the Director and the Console. In addition, if the console name, provided on the **Name =** directive, is the same as a Client name, the user of that console is permitted to use the **SetIP** command to change the Address directive in the Director's client resource to the IP address of the Console. This permits portables or other machines using DHCP (non-fixed IP addresses) to "notify" the Director of their current IP address.

The Console resource is optional and need not be specified. However, if it is specified, you can use ACLs (Access Control Lists) in the Director's configuration file to restrict the particular console (or user) to see only information pertaining to his jobs or client machine.

You may specify as many Console resources in the console's conf file. If you do so, generally the first Console resource will be used. However, if you have multiple Director resources (i.e. you want to connect to different directors), you can bind one of your Console resources to a particular Director resource, and thus when you choose a particular Director, the appropriate Console configuration resource will be used. See the "Director" directive in the Console resource described below for more information.

Note, the Console resource is optional, but can be useful for restricted consoles as noted above.

Console Start of the Console resource.

Name = <name> The Console name used to allow a restricted console to change its IP address using the SetIP command. The SetIP command must also be defined in the Director's conf CommandACL list.

Password = <password> If this password is supplied, then the password specified in the Director resource of you Console conf will be ignored. See below for more details.

Director = <director-resource-name> If this directive is specified, this Console resource will be used by bconsole when that particular director is selected when first starting bconsole. I.e. it binds a particular console resource with its name and password to a particular director.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Console to set a keepalive interval (heartbeat) in seconds on each of the sockets to communicate with the Director. It is implemented only on systems (Linux, ...) that provide the **setsockopt** TCP_KEEPIDLE function. The default value is zero, which means no change is made to the socket.

The following configuration files were supplied by Phil Stracchino. For example, if we define the following in the user's bconsole.conf file (or perhaps the bwx-console.conf file):

```
Director {
   Name = MyDirector
   DIRport = 9101
   Address = myserver
   Password = "XXXXXXXXXXX"    # no, really.  this is not obfuscation.
}


Console {
   Name = restricted-user
   Password = "UntrustedUser"
}
```

Where the Password in the Director section is deliberately incorrect, and the Console resource is given a name, in this case **restricted-user**. Then in the Director's bacula-dir.conf file (not directly accessible by the user), we define:

```
Console {
  Name = restricted-user
  Password = "UntrustedUser"
  JobACL = "Restricted Client Save"
  ClientACL = restricted-client
  StorageACL = main-storage
  ScheduleACL = *all*
  PoolACL = *all*
  FileSetACL = "Restricted Client's FileSet"
  CatalogACL = DefaultCatalog
  CommandACL = run
}
```

the user logging into the Director from his Console will get logged in as **restricted-user**, and he will only be able to see or access a Job with the name **Restricted Client Save** a Client with the name **restricted-client**, a Storage device **main-storage**, any Schedule or Pool, a FileSet named **Restricted Client's FileSet**, a Catalog named **DefaultCatalog**, and the only command he can use in the Console is the **run** command. In other words, this user is rather limited in what he can see and do with Bacula.

The following is an example of a bconsole conf file that can access several Directors and has different Consoles depending on the director:

```
Director {
   Name = MyDirector
   DIRport = 9101
   Address = myserver
   Password = "XXXXXXXXXXX"    # no, really.  this is not obfuscation.
}

Director {
   Name = SecondDirector
   DIRport = 9101
   Address = secondserver
   Password = "XXXXXXXXXXX"    # no, really.  this is not obfuscation.
}

Console {
   Name = restricted-user
   Password = "UntrustedUser"
   Director = MyDirector
}

Console {
   Name = restricted-user
   Password = "A different UntrustedUser"
   Director = SecondDirector
}
```

The second Director referenced at "secondserver" might look like the following:

```
Console {
  Name = restricted-user
  Password = "A different UntrustedUser"
  JobACL = "Restricted Client Save"
  ClientACL = restricted-client
  StorageACL = second-storage
  ScheduleACL = *all*
  PoolACL = *all*
  FileSetACL = "Restricted Client's FileSet"
  CatalogACL = RestrictedCatalog
  CommandACL = run, restore
  WhereACL = "/"
}
```

10.5 Console Commands

For more details on running the console and its commands, please see the Bacula Console chapter of this manual.

10.6 Sample Console Configuration File

An example Console configuration file might be the following:

```
#
# Bacula Console Configuration File
#
Director {
  Name = HeadMan
  address = "my_machine.my_domain.com"
  Password = Console_password
}
```

Chapter 11

Monitor Configuration

The Monitor configuration file is a stripped down version of the Director configuration file, mixed with a Console configuration file. It simply contains the information necessary to contact Directors, Clients, and Storage daemons you want to monitor.

For a general discussion of configuration file and resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual.

The following Monitor Resource definition must be defined:

- Monitor – to define the Monitor's name used to connect to all the daemons and the password used to connect to the Directors. Note, you must not define more than one Monitor resource in the Monitor configuration file.
- At least one Client, Storage or Director resource, to define the daemons to monitor.

11.1 The Monitor Resource

The Monitor resource defines the attributes of the Monitor running on the network. The parameters you define here must be configured as a Director resource in Clients and Storages configuration files, and as a Console resource in Directors configuration files.

Monitor Start of the Monitor records.

Name = <name> Specify the Director name used to connect to Client and Storage, and the Console name used to connect to Director. This record is required.

Password = <password> Where the password is the password needed for Directors to accept the Console connection. This password must be identical to the **Password** specified in the **Console** resource of the Director's configuration file. This record is required if you wish to monitor Directors.

Refresh Interval = <time> Specifies the time to wait between status requests to each daemon. It can't be set to less than 1 second, or more than 10 minutes, and the default value is 5 seconds.

11.2 The Director Resource

The Director resource defines the attributes of the Directors that are monitored by this Monitor.

As you are not permitted to define a Password in this resource, to avoid obtaining full Director privileges, you must create a Console resource in the Director's configuration file, using the Console Name and Password defined in the Monitor resource. To avoid security problems, you should configure this Console resource to allow access to no other daemons, and permit the use of only two commands: **status** and **.status** (see below for an example).

You may have multiple Director resource specifications in a single Monitor configuration file.

Director Start of the Director records.

Name = <name> The Director name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Director's configuration file. This record is required.

DIRPort = <port-number> Specify the port to use to connect to the Director. This value will most likely already be set to the value you specified on the **--with-base-port** option of the **./configure** command. This port must be identical to the **DIRport** specified in the **Director** resource of the Director's configuration file. The default is 9101 so this record is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director. This record is required.

11.3 The Client Resource

The Client resource defines the attributes of the Clients that are monitored by this Monitor.

You must create a Director resource in the Client's configuration file, using the Director Name defined in the Monitor resource. To avoid security problems, you should set the **Monitor** directive to **Yes** in this Director resource.

You may have multiple Director resource specifications in a single Monitor configuration file.

Client (or FileDaemon) Start of the Client records.

Name = <name> The Client name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Client's configuration file. This record is required.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address in dotted quad notation for a Bacula File daemon. This record is required.

FD Port = <port-number> Where the port is a port number at which the Bacula File daemon can be contacted. The default is 9102.

Password = <password> This is the password to be used when establishing a connection with the File services, so the Client configuration file on the machine to be backed up must have the same password defined for this Director. This record is required.

11.4 The Storage Resource

The Storage resource defines the attributes of the Storages that are monitored by this Monitor.

You must create a Director resource in the Storage's configuration file, using the Director Name defined in the Monitor resource. To avoid security problems, you should set the **Monitor** directive to **Yes** in this Director resource.

You may have multiple Director resource specifications in a single Monitor configuration file.

Storage Start of the Storage records.

Name = <name> The Storage name used to identify the Director in the list of monitored daemons. It is not required to be the same as the one defined in the Storage's configuration file. This record is required.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address in dotted quad notation for a Bacula Storage daemon. This record is required.

SD Port = <port> Where port is the port to use to contact the storage daemon for information and to start jobs. This same port number must appear in the Storage resource of the Storage daemon's configuration file. The default is 9103.

Password = <password> This is the password to be used when establishing a connection with the Storage services. This same password also must appear in the Director resource of the Storage daemon's configuration file. This record is required.

11.5 Tray Monitor Security

There is no security problem in relaxing the permissions on tray-monitor.conf as long as FD, SD and DIR are configured properly, so the passwords contained in this file only gives access to the status of the daemons. It could be a security problem if you consider the status information as potentially dangerous (I don't think it is the case).

Concerning Director's configuration:
In tray-monitor.conf, the password in the Monitor resource must point to a restricted console in bacula-dir.conf (see the documentation). So, if you use this password with bconsole, you'll only have access to the status of the director (commands status and .status). It could be a security problem if there is a bug in the ACL code of the director.

Concerning File and Storage Daemons' configuration:
In tray-monitor.conf, the Name in the Monitor resource must point to a Director resource in bacula-fd/sd.conf, with the Monitor directive set to Yes (once again, see the documentation). It could be a security problem if there is a bug in the code which check if a command is valid for a Monitor (this is very unlikely as the code is pretty simple).

11.6 Sample Tray Monitor configuration

An example Tray Monitor configuration file might be the following:

```
#
# Bacula Tray Monitor Configuration File
#
Monitor {
  Name = rufus-mon        # password for Directors
  Password = "GN0uRo7PTUmlMbqrJ2Gr1p0fk0HQJTxwnFyE4WSST3MWZseR"
  RefreshInterval = 10 seconds
}

Client {
  Name = rufus-fd
  Address = rufus
  FDPort = 9102           # password for FileDaemon
  Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
}
Storage {
  Name = rufus-sd
  Address = rufus
  SDPort = 9103           # password for StorageDaemon
  Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
}
Director {
  Name = rufus-dir
```

```
  DIRport = 9101
  address = rufus
}
```

11.6.1 Sample File daemon's Director record.

Click here to see the full example.

```
#
# Restricted Director, used by tray-monitor to get the
#   status of the file daemon
#
Director {
  Name = rufus-mon
  Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZ0bxT3XQxgxppTn"
  Monitor = yes
}
```

11.6.2 Sample Storage daemon's Director record.

Click here to see the full example.

```
#
# Restricted Director, used by tray-monitor to get the
#   status of the storage daemon
#
Director {
  Name = rufus-mon
  Password = "9usxgc307dMbe7jbD16v0PXlhD64UVasIDD0DH2WAujcDsc6"
  Monitor = yes
}
```

11.6.3 Sample Director's Console record.

Click here to see the full example.

```
#
# Restricted console used by tray-monitor to get the status of the director
#
Console {
  Name = Monitor
  Password = "GN0uRo7PTUmlMbqrJ2Gr1p0fk0HQJTxwnFyE4WSST3MWZseR"
  CommandACL = status, .status
}
```

Chapter 12

Bacula Security Issues

- Security means being able to restore your files, so read the Critical Items Chapter of this manual.
- The Clients (**bacula-fd**) must run as root to be able to access all the system files.
- It is not necessary to run the Director as root.
- It is not necessary to run the Storage daemon as root, but you must ensure that it can open the tape drives, which are often restricted to root access by default. In addition, if you do not run the Storage daemon as root, it will not be able to automatically set your tape drive parameters on most OSes since these functions, unfortunately require root access.
- You should restrict access to the Bacula configuration files, so that the passwords are not world-readable. The **Bacula** daemons are password protected using CRAM-MD5 (i.e. the password is not sent across the network). This will ensure that not everyone can access the daemons. It is a reasonably good protection, but can be cracked by experts.
- If you are using the recommended ports 9101, 9102, and 9103, you will probably want to protect these ports from external access using a firewall and/or using tcp wrappers (**etc/hosts.allow**).
- By default, all data that is sent across the network is unencrypted. However, Bacula does support TLS (transport layer security) and can encrypt transmitted data. Please read the TLS (SSL) Communications Encryption section of this manual.
- You should ensure that the Bacula working directories are readable and writable only by the Bacula daemons.
- If you are using **MySQL** it is not necessary for it to run with **root** permission.
- The default Bacula **grant-mysql-permissions** script grants all permissions to use the MySQL database without a password. If you want security, please tighten this up!
- Don't forget that Bacula is a network program, so anyone anywhere on the network with the console program and the Director's password can access Bacula and the backed up data.
- You can restrict what IP addresses Bacula will bind to by using the appropriate **DirAddress**, **FDAddress**, or **SDAddress** records in the respective daemon configuration files.
- Be aware that if you are backing up your database using the default script, if you have a password on your database, it will be passed as a command line option to that script, and any user will be able to see this information. If you want it to be secure, you will need to pass it by an environment variable or a secure file.

  See also Backing Up Your Bacula Database - Security Considerations for more information.

12.1 Backward Compatibility

One of the major goals of Bacula is to ensure that you can restore tapes (I'll use the word tape to include disk Volumes) that you wrote years ago. This means that each new version of Bacula should be able to read old format tapes. The first problem you will have is to ensure that the hardware is still working some years down the road, and the second problem will be to ensure that the media will still be good, then your OS must be able to interface to the device, and finally Bacula must be able to recognize old formats. All the problems except the last are ones that we cannot solve, but by careful planning you can.

Since the very beginning of Bacula (January 2000) until today (December 2005), there have been two major Bacula tape formats. The second format was introduced in version 1.27 in November of 2002, and it has not changed since then. In principle, Bacula can still read the original format, but I haven't tried it lately so who knows ...

Though the tape format is fixed, the kinds of data that we can put on the tapes are extensible, and that is how we added new features such as ACLs, Win32 data, encrypted data, ... Obviously, an older version of Bacula would not know how to read these newer data streams, but each newer version of Bacula should know how to read all the older streams.

If you want to be 100should:

1. Try reading old tapes from time to time – e.g. at least once a year.

2. Keep statically linked copies of every version of Bacula that you use in production then if for some reason, we botch up old tape compatibility, you can always pull out an old copy of Bacula ...

The second point is probably overkill but if you want to be sure, it may save you someday.

12.2 Configuring and Testing TCP Wrappers

TCP Wrappers are implemented if you turn them on when configuring (**./configure --with-tcp-wrappers**). With this code enabled, you may control who may access your daemons. This control is done by modifying the file: **/etc/hosts.allow**. The program name that **Bacula** uses when applying these access restrictions is the name you specify in the daemon configuration file (see below for examples). You must not use the **twist** option in your **/etc/hosts.allow** or it will terminate the Bacula daemon when a connection is refused.

The exact name of the package you need loaded to build with TCP wrappers depends on the system. For example, on SuSE, the TCP wrappers libraries needed to link Bacula are contained in the tcpd-devel package. On Red Hat, the package is named tcp_wrappers.

Dan Langille has provided the following information on configuring and testing TCP wrappers with Bacula.

If you read hosts_options(5), you will see an option called twist. This option replaces the current process by an instance of the specified shell command. Typically, something like this is used:

```
ALL : ALL \
 : severity auth.info \
 : twist /bin/echo "You are not welcome to use %d from %h."
```

The libwrap code tries to avoid **twist** if it runs in a resident process, but that test will not protect the first hosts_access() call. This will result in the process (e.g. bacula-fd, bacula-sd, bacula-dir) being terminated if the first connection to their port results in the twist option being invoked. The potential, and I stress potential, exists for an attacker to prevent the daemons from running. This situation is eliminated if your /etc/hosts.allow file contains an appropriate rule set. The following example is sufficient:

```
undef-fd : localhost : allow
undef-sd : localhost : allow
```

```
undef-dir : localhost : allow
undef-fd : ALL : deny
undef-sd : ALL : deny
undef-dir : ALL : deny
```

You must adjust the names to be the same as the Name directives found in each of the daemon configuration files. They are, in general, not the same as the binary daemon names. It is not possible to use the daemon names because multiple daemons may be running on the same machine but with different configurations.

In these examples, the Director is undef-dir, the Storage Daemon is undef-sd, and the File Daemon is undef-fd. Adjust to suit your situation. The above example rules assume that the SD, FD, and DIR all reside on the same box. If you have a remote FD client, then the following rule set on the remote client will suffice:

```
undef-fd : director.example.org : allow
undef-fd : ALL : deny
```

where director.example.org is the host which will be contacting the client (ie. the box on which the Bacula Director daemon runs). The use of "ALL : deny" ensures that the twist option (if present) is not invoked. To properly test your configuration, start the daemon(s), then attempt to connect from an IP address which should be able to connect. You should see something like this:

```
$ telnet undef 9103
Trying 192.168.0.56...
Connected to undef.example.org.
Escape character is '^]'.
Connection closed by foreign host.
$
```

This is the correct response. If you see this:

```
$ telnet undef 9103
Trying 192.168.0.56...
Connected to undef.example.org.
Escape character is '^]'.
You are not welcome to use undef-sd from xeon.example.org.
Connection closed by foreign host.
$
```

then twist has been invoked and your configuration is not correct and you need to add the deny statement. It is important to note that your testing must include restarting the daemons after each connection attempt. You can also tcpdchk(8) and tcpdmatch(8) to validate your /etc/hosts.allow rules. Here is a simple test using tcpdmatch:

```
$ tcpdmatch undef-dir xeon.example.org
warning: undef-dir: no such process name in /etc/inetd.conf
client: hostname xeon.example.org
client: address 192.168.0.18
server: process undef-dir
matched: /etc/hosts.allow line 40
option: allow
access: granted
```

If you are running Bacula as a standalone daemon, the warning above can be safely ignored. Here is an example which indicates that your rules are missing a deny statement and the twist option has been invoked.

```
$ tcpdmatch undef-dir 10.0.0.1
warning: undef-dir: no such process name in /etc/inetd.conf
client: address 10.0.0.1
server: process undef-dir
matched: /etc/hosts.allow line 91
option: severity auth.info
option: twist /bin/echo "You are not welcome to use
  undef-dir from 10.0.0.1."
access: delegated
```

12.3 Running as non-root

Security advice from Dan Langille:

It is a good idea to run daemons with the lowest possible privileges. In other words, if you can, don't run applications as root which do not have to be root. The Storage Daemon and the Director Daemon do not need to be root. The File Daemon needs to be root in order to access all files on your system. In order to run as non-root, you need to create a user and a group. Choosing `bacula` as both the user name and the group name sounds like a good idea to me.

The FreeBSD port creates this user and group for you. Here is what those entries looked like on my FreeBSD laptop:

```
bacula:*:1002:1002::0:0:Bacula Daemon:/var/db/bacula:/sbin/nologin
```

I used vipw to create this entry. I selected a User ID and Group ID of 1002 as they were unused on my system.

I also created a group in /etc/group:

```
bacula:*:1002:
```

The bacula user (as opposed to the Bacula daemon) will have a home directory of `/var/db/bacula` which is the default location for the Bacula database.

Now that you have both a bacula user and a bacula group, you can secure the bacula home directory by issuing this command:

```
chown -R bacula:bacula /var/db/bacula/
```

This ensures that only the bacula user can access this directory. It also means that if we run the Director and the Storage daemon as bacula, those daemons also have restricted access. This would not be the case if they were running as root.

It is important to note that the storage daemon actually needs to be in the operator group for normal access to tape drives etc (at least on a FreeBSD system, that's how things are set up by default) Such devices are normally chown root:operator. It is easier and less error prone to make Bacula a member of that group than it is to play around with system permissions.

Starting the Bacula daemons

To start the bacula daemons on a FreeBSD system, issue the following command:

```
/usr/local/etc/rc.d/bacula-dir start
/usr/local/etc/rc.d/bacula-sd  start
/usr/local/etc/rc.d/bacula-fd  start
```

To confirm they are all running:

```
$ ps auwx | grep bacula
root   63418 0.0 0.3 1856 1036 ?? Ss 4:09PM 0:00.00
    /usr/local/sbin/bacula-fd -v -c /usr/local/etc/bacula-fd.conf
bacula 63416 0.0 0.3 2040 1172 ?? Ss 4:09PM 0:00.01
    /usr/local/sbin/bacula-sd -v -c /usr/local/etc/bacula-sd.conf
bacula 63422 0.0 0.4 2360 1440 ?? Ss 4:09PM 0:00.00
    /usr/local/sbin/bacula-dir -v -c /usr/local/etc/bacula-dir.conf
```

General Index

Director Index

File Daemon Index

Storage Daemon Index

Console Index