Bacula® Storage Management System

The Bacula® Storage Management System Version 1.29

It comes by night and sucks the vital essence from your computers

---

This document was last updated 22 January 2003.

If you are viewing this document with Netscape, it probably looks really poor.
I recommend that you use Galeon or Mozilla instead.

General Documents

Other Notes

Bugs

Bacula Design Documents

What is Bacula?

Bacula is a set of computer programs that permit you (or the system administrator) to manage backup, recovery, and verification of computer data across a network of computers of different kinds. In technical terms, it is a network Client/Server based backup program. Bacula is relatively easy to use and efficient, while offering many advanced storage management features that make it easy to find and recover lost or damaged files. Due to its modular design, Bacula is scalable from small single computer systems to systems consisting of hundreds of computers located over a large network.

Who Needs Bacula?

If you are currently using a program such as **tar**, **dump**, or **bru** to backup your computer data, and you would like a network solution, more flexibility, or catalog services, **Bacula** will most likely provide the additional features you want.

If you are running **Amanda** and would like a backup program that can write to multiple volumes (i.e. is not limited by your tape drive capacity), **Bacula** can most likely fill your needs.

If you are currently using a sophisticated commercial package such as Legato Networker. ARCserveIT, Arkeia, or PerfectBackup+, you may be interested in Bacula, which provides many of the same features, and is free software available under the GNU Version 2 software license.

Bacula Components or Services

Bacula is made up of the following five major components or services:

- **Bacula Director** service consists of the program that supervises all the backup, restore, verify and archive operations. The system administrator uses the Bacula Director to schedule backups and to recover files. For more details see the Director Services Daemon Design Document. The Director runs as a daemon or a service (i.e. in the background).
- **Bacula Console** services is the program that allows the administrator or user to communicate with the **Bacula Director** (see above). Currently, the Bacula Console is available in two versions. The first and simplest is to run the Console program in a shell window (i.e. TTY interface). Most system administrators will find this completely adequate. The second version is a GNOME GUI interface that for the moment (26 December 2002) is far from complete, but quite functional as it has all the capabilities of the shell Console. For more details see the Bacula Console Design Document.
- **Bacula File** services (or Client program) is the software program that is installed on the machine to be backed up. It is specific to the operating system on which it runs and is responsible for providing the file attributes and data when requested by the Director. The File services are also responsible for the file system dependent part of restoring the file attributes and data during a recovery operation. For more details see the File Services Daemon Design Document. This program runs as a daemon on the machine to be backed up, and in some of the documentation, the File daemon is referred to as the Client (for example in Bacula's configuration file). In addition to Unix/Linux File daemons, there is a Windows File daemon (normally distributed as in binary format). The Windows File daemon runs on all currently known Windows versions (95, 98, Me, NT, 2000, XP).

- **Bacula Storage** services consist of the software programs that perform the storage and recovery of the file attributes and data to the physical backup media or volumes. In other words, the Storage daemon is responsible for reading and writing your tapes (or other storage media, e.g. files). For more details see the Storage Services Daemon Design Document. The Storage services runs as a daemon on the machine that has the backup device (usually a tape drive).
- **Catalog** services are comprised of the software programs responsible for maintaining the file indexes and volume databases for all files backed up. The Catalog services permit the System Administrator or user to quickly locate and restore any desired file. The Catalog services of **Bacula** set it apart from programs like tar and bru, since the catalog maintains a record of all Volumes used, all Jobs run, and all Files saved. **Bacula** currently supports three different databases, MySQL, SQLite, and an internal database, one of which must be chosen when building **Bacula**.

  The internal Bacula database is simple but does not provide the full set of features and queries available with a SQL database. As a consequence, we highly recommend using one of the two SQL databases currently supported (MySQL or SQLite) since they provide many more features, including rapid indexing, arbitrary queries, and security. Although we plan to support other major SQL databases, the current **Bacula** implementation currently interfaces only to MySQL and SQLite. For more details see the Catalog Services Design Document.

  The RPMs for MySQL ship as part of the Linux RedHat release, or building it from the source is quite easy, see the Installing and Configuring MySQL chapter of this document for the details. For more information on MySQL, please see: www.mysql.com.

  Configuring and building SQLite is even easier. For the details of configuring SQLite, please see the Installing and Configuring SQLite chapter of this document.

To perform a successful save or restore, the following four daemons must be configured and running: the Director daemon, the File daemon, the Storage daemon, and MySQL or SQLite.

Conventions Used in this Document

Bacula is in a state of evolution, and as a consequence, this manual will not always agree with the code. If an item in this manual is preceded by an asterisk (*), it indicates that the particular feature is not implemented. If it is preceded by a plus sign (+), it indicates that the feature may be partially implemented.

If you are reading this manual as supplied in a released version of the software, the above paragraph holds true. If you are reading the online version of the manual, www.bacula.org/manual, please bear in mind that this version describes the current version in development (in the CVS), but generally lags behind the code a bit.

Quick Start

To get **Bacula** up and running quickly, we recommend that you first scan the Terminology section below, then quickly review the next chapter entitled The Current State of Bacula, then the Quick Start Guide to Bacula, which will give you a quick overview of getting **Bacula** running. After which, you should proceed to the chapter on Installing Bacula, then How to Configure

Bacula, and finally the chapter on Running Bacula.

Terminology

To facilitate communication about this project, we provide here the definitions of the terminology that we use.

Administrator
: The person or persons responsible for administrating the Bacula system.

Backup
: We use the term **Backup** to refer to a **Bacula** Job that saves files.

Catalog
: The Catalog is used to store summary information about the Jobs, Clients, and Files that were backed up and on what Volume or Volumes. It permits the administrator or user to determine what jobs were run and their status as well as the important characteristics of each file that was backed up. The Catalog is an online resource, and does not contain the data for the files backed up. Most of the information stored in the catalog is also stored on the backup volumes (i.e. tapes). Of course, the tapes will also have a copy of the file in addition to the File Attributes (see below).
The catalog feature is one part of **Bacula** that distinguishes it from simple backup and archive programs such as **dump** and **tar**.

Client
: In Bacula's terminology, the word Client refers to the machine being backed up, and it is synonymous with the File services or File daemon.

Console
: The program that interfaces to the Director allowing the user or system administrator to control Bacula.

Daemon
: Unix terminology for a program that is always present in the background to carry out a designated task. On Windows systems, as well as some Linux systems, daemons are called **Services**.

Director
: The main Bacula server daemon that schedules and directs all Bacula operations.

Differential
: A backup that includes all files changed since the last Full save started. Note, other backup programs may define this differently.

File Attributes
: The File Attributes are all the information necessary about a file to identify it and all its properties such as size, creation date, modification date, permissions, etc. Normally, the attributes are handled entirely by **Bacula** so that the user never needs to be concerned about them. The attributes do not include the file's data.

File Service or File Daemon
: The daemon running on the client computer to be backed up. This is also referred to as the File services, and sometimes as the Client services.

FileSet
: Defines the files to be backed up. It consists of a list of included files or directories, a list of excluded files, and how the file is to be stored (compression, encryption, signatures). For more details, see the FileSet Resource definition in the Director chapter of this document.

Incremental

A backup that includes all files changed since the last Full, Differential, or Incremental backup started.

Job

A Bacula Job defines the work that Bacula must perform to backup or restore a particular Client. It consists of the **Type** (backup, restore, verify, etc), the **Level** (full, incremental,...), the **FileSet**, and **Storage** the files are to be backed up (Storage device, Media Pool). For more details, see the Job Resource definition in the Director chapter of this document.

Restore

A restore is the operation of recovering a file (lost or damaged) from a backup medium. It is the inverse of a save, except that in most cases, a restore will normally have a small set of files to restore, while normally a Save backs up all the files on the system. Of course, after a disk crash, a **Bacula** can be called upon to do a full Restore of all files that were on the system.

Schedule

Defines when the Bacula Job will be scheduled for execution. For more details, see the Schedule Resource definition in the Director chapter of this document.

Service

This is Windows terminology for a **daemon** −− see above. It is now commonly used in Unix environments as well.

Storage Coordinates

The information returned from the Storage Services that uniquely locates a file on a backup medium. It consists of two parts: one part pertains to each file saved, and the other part pertains to the whole Job. Normally, this information is saved in the Catalog so that the user doesn't need specific knowledge of the Storage Coordinates. The Storage Coordinates include the File Attributes (see above) plus the unique location of the information on the backup Volume.

Session

Normally refers to the internal conversation between the File daemon and the Storage daemon. The File daemon opens a **session** with the Storage daemon to save a FileSet, or to restore it. A session has a one to one correspondence to a Bacula Job (see above).

Verify

A verify is a job that compares the current file attributes to the attributes that have previously been stored in the Bacula Catalog. This feature can be used for detecting changes to critical system files similar to what **Tripwire** does. One of the major advantages of using **Bacula** to do this is that on the machine you want protected such as a server, you can run just the File daemon, and the Director, Storage daemon, and Catalog reside on a different machine. As a consequence, if your server is ever compromised, it is unlikely that your verification database will be tampered with. Verify can also be used to check that the most recent Job data written to a Volume agrees with what is stored in the Catalog (i.e. it compares the file attributes), *or it can check the Volume contents against the original files on disk.

**Archive*

An Archive operation is done after a Save, and it consists of removing the Volumes on which data is saved from active use. These Volumes are marked as Archived, and many no longer be used to save files. All the files contained on an Archived Volume are removed from the Catalog. NOT YET IMPLEMENTED.

**Update*

An Update operation causes the files on the remote system to be updated to be the same as the host system. This is equivalent to an **rdist** capability. NOT YET

IMPLEMENTED.

Retention Period

There are various kinds of retention periods that **Bacula** recognizes. The most important are the **File** Retention Period, **Job** Retention Period, and the **Volume** Retention Period. Each of these retention periods applies to the time that specific records will be kept in the Catalog database. This should not be confused with the time that the data saved to a Volume is valid.
The File Retention Period determines the time that File records are kept in the catalog database. This period is important because the volume of the database File records by far use the most storage space in the database. As a consequence, you must ensure that regular "pruning" of the database file records are done. (See the Console **retention** command for more details on this subject).

The Job Retention Period is the length of time that Job records will be kept in the database. Note, all the File records are tied to the Job that saved those files. The File records can be purged leaving the Job records. In this case, information will be available about the jobs that ran, but not the details of the files that were backed up. Normally, when a Job record is purged, all its File records will also be purged.

The Volume Retention Period is the minimum of time that a Volume will be kept before it is reused. **Bacula** will normally never overwrite a Volume that contains the only backup copy of a file. Under ideal conditions, the Catalog would retain entries for all files backed up for all current Volumes. Once a Volume is overwritten, the files that were backed up on that Volume are automatically removed from the Catalog. However, if there is a very large pool of Volumes or a Volume is never overwritten, the Catalog database may become enormous. To keep the Catalog to a manageable size, the backup information should removed from the Catalog after the defined File Retention Period. **Bacula** provides the mechanisms for the catalog to be automatically pruned according to the retention periods defined.

Scan

A Scan operation causes the contents of a Volume or a series of Volumes to be scanned. These Volumes with the information on which files they contain are restored to the Bacula Catalog. Once the information is restored to the Catalog, the files contained on those Volumes may be easily restored. This function is particularly useful if certain Volumes or Jobs have gone past (that is their retention period and have been pruned or purged from the Catalog. Scanning data from Volumes into the Catalog is done by using the **bscan** program. See the bscan section of the Bacula Utilities Chapter of this manual for more details.

What Bacula is Not

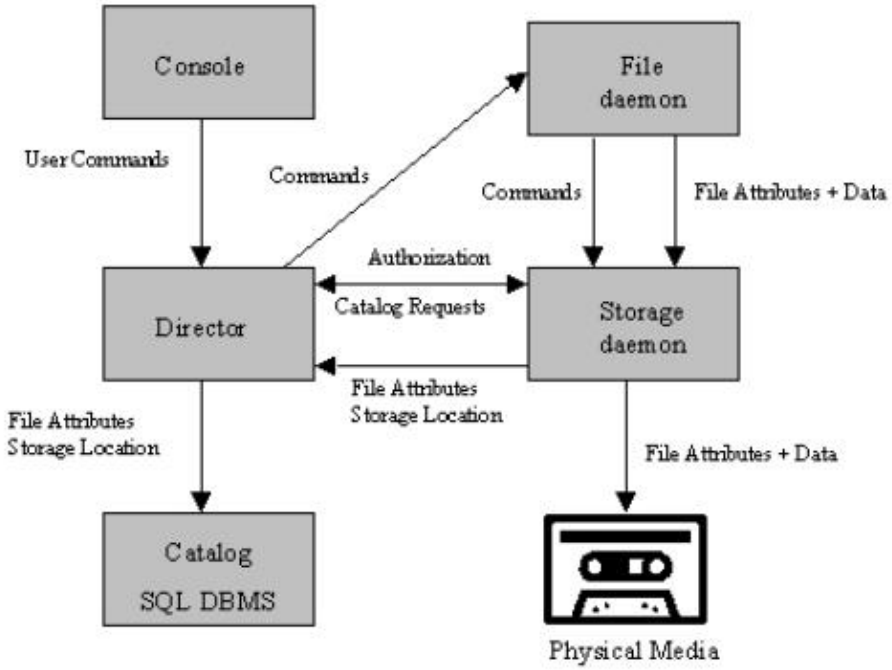
Bacula is a backup, restore and verification program and is not a complete disaster recovery system unless you plan carefully and follow the instructions included in the Disaster Recovery Chapter of this manual.

With proper planning, as mentioned in the Disaster Recovery chapter **Bacula** can be used the key component of your disaster recovery system. For example, if you have created an emergency boot disk, a Bacula Rescue disk to save the current partitioning information of your hard disk, and maintain a complete Bacula backup, it is possible to completely recover your system from "bare metal".

If you have used the **WriteBootstrap** record in your job or some other means to save a valid bootstrap file, you will be able to use it to extract the necessary files (without using the catalog or manually searching for the files to restore).

Interactions Between the Bacula Services

The following block diagram shows the typical interactions between the Bacula Services for a backup job. Each block represents in general a separate process (normally a daemon). In general, the Director oversees the flow of information. It also maintains the Catalog.



Index Features

Introduction Index Getting Started

The Current State of Bacula

In other words, what is and what is not currently implemented and functional.

What is Implemented

- Internal scheduler for automatic Job execution.
- Scheduling of multiple Jobs at the same time.
- You may run one Job at a time or multiple simultaneous Jobs. Multiple simultaneous jobs is not currently recommended.
- Restore of one or more files selected interactively.
- Restore of a complete system starting from bare metal. This is mostly automated for Linux systems and partially automated for Solaris. See Disaster Recovery Using Bacula.
- Listing and Restoration of files using stand−alone **bls** and **bextract** tool programs. Among other things, this permits extraction of files when **Bacula** and/or the catalog are not available.
- Console interface to the Director allowing complete control. Both a shell and GNOME GUI versions of the Console program are available.
- Verification of files previously cataloged, permitting a Tripwire like capability (system break−in detection).
- CRAM−MD5 password Authentication between each component (daemon).
- A comprehensive and extensible configuration file for each daemon.
- Catalog database facility for remembering Volumes, Pools, Jobs, and Files backed up.
- Support for SQLite and MySQL Catalog databases.
- User extensible queries to the SQLite and MySQL databases.
- Simple Internal Database if SQLite or MySQL is not available (mainly for developers).
- Labeled Volumes, preventing accidental overwriting (at least by Bacula).
- Any number of Jobs and Clients can be backed up to a single Volume. That is, you can backup and restore Linux, Unix, Sun, and Windows machines to the same Volume.
- Multi−volume saves. When a Volume is full, **Bacula** automatically requests the next Volume and continues the backup.
- Pool and Volume management providing Volume flexibility (e.g. monthly, weekly, daily Volume sets, Volume sets segregated by Client, ...).
- Simple shell interface support for virtually any autoloader program. Several autoloaders including **mtx** are currently implemented.
- Machine independent Volume data format. Linux, Solaris, and Windows clients can all be backed up to the same Volume if desired.
- A flexible message handler including routing of messages from any daemon back to the Director.
- Multi−threaded implementation.
- Mechanisms to handle arbitrarily long filenames and messages.
- GZIP Compression on a file by file basis.

Advantages of Bacula Over Other Backup Programs

- Since there is a client for each machine, you can backup and restore clients of any type ensuring that all attributes of files are properly saved and restored.
- It is also possible to backup clients without any client software by using NFS or Samba. However, if possible, we recommend running a Client File daemon on each machine to

be backed up.
- Bacula handles multi−volume backups.
- A full comprehensive database of all files backed up. This permits online viewing of files saved on any particular Volume.
- Automatic pruning of the database (removal of old records) thus simplifying database administration.
- Any SQL database engine can be used making Bacula very flexible.
- The modular but integrated design makes **Bacula** very scalable.
- Since Bacula uses client file servers, any database or other application can be properly shutdown by **Bacula** using the native tools of the system, backed up, then restarted (all within a **Bacula** Job).
- Bacula has a built−in Job scheduler.
- The Volume format is documented and there are simple C programs to read/write it.
- Bacula uses well defined (registered) ports −− no rpcs, no shared memory.
- Bacula installation and configuration is relatively simple compared to other comparable products.

Current Implementation Weaknesses

- The graphical user interface is currently in an infant stage.
- Currently (as of version 1.28) we recommend that you not run multiple simultaneous jobs backing up to the same Volume. This is because the restore code is not yet mature enough (tested enough) to deal with intermingled blocks from multiple simultaneous jobs.
- Typical of Microsoft, not all files can always be saved on WinNT and Win2000. Anyone knowing the magic incantations please step forward. The files that are skipped seem to be in exclusive use by some other process, and don't appear to be too important.

Other Items Not Implemented (but planned)

- Complete error checking on configuration files.
- Event handlers are not yet implemented (e.g. when Job terminates do this, ...)
- File System Modules (configurable routines for saving/restoring special files).
- Data encryption between the daemons.

Temporary Limitations or Restrictions

- All three daemons (DIR, FD, SD) must be running for a Job to run. If you use MySQL as the catalog, it must also be running. If you use SQLite as the catalog, it will be started automatically. This isn't very significant as most other programs have the same limitation.
- Unicode is not yet supported.
- 128 bit integers are not yet implemented.
- Supports only IPv4.

Design Limitations or Restrictions

- Names (resource names, Volume names, and such) defined in Bacula configuration files are limited to a fixed number of characters. Currently the limit is defined as 127

characters. Note, this does not apply to filenames, which may be arbitrarily long.
- There is no concept of a Pool of backup devices (i.e. if device /dev/nst0 is busy, use /dev/nst1, ...).


Introduction

Index

Getting Started


The Current State of Bacula

Index

Installing Bacula

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.

Supported Systems

- Linux systems (built and tested on RedHat).
- Solaris
- FreeBSD (note, although Bacula runs quite very reliably on FreeBSD systems, there have been some reports that Bacula does not work with the FreeBSD SCSI tape driver)
- Windows Client (File daemon) binaries
- Irix File daemon (implemented but production testing not confirmed)
- See the Porting Chapter of this manual for information on porting to other systems.

Requirements

- **Bacula** has been compiled and run on Linux RedHat, FreeBSD, and Solaris systems.
- It requires GNU C++ version 2.96 to compile. You can try with other compilers and other version, but you are on your own. (GCC 3.2 not recommended at this time.) Note, in general GNU C++ is a separate package (e.g. RPM) from GNU C, so you need them both loaded.
- There are certain third party packages that **Bacula** needs. Except for MySQL, they can all be found in the **depkgs** and **depkgs1** releases.
- If you want to build the Win32 binaries, you will need the full Cygwin 1.3.10 release. Although all components build (console has some warnings), only the File daemon has been tested. Please note that if you attempt to build Bacula on any other version of Cygwin, particularly older versions or with GCC 3.2, you will be on your own.
- **Bacula** requires a good implementation of pthreads to work.
- The source code has been written with portability in mind and is mostly POSIX compatible. Thus porting to any POSIX compatible operating system should be relatively easy.

Up Front Decisions

- Before building **Bacula** you need to decide if you want to use SQLite, MySQL or the internal Bacula database. Unless you are already familiar with MySQL, we suggest that you use the SQLite database as it is the simplest. If you need security or have a large operation, you should consider the MySQL database because it provides all the **Bacula** features and is the best tested. MySQL is also much easier to upgrade than SQLite when new versions of **Bacula** require a database update.
- If you wish to use SQLite as the Bacula catalog, please see Installing and Configuring SQLite chapter of this manual.
- If you wish to use MySQL as the Bacula catalog, please see the Installing and Configuring MySQL chapter of this manual.
- If you do choose to use the internal database, you need do nothing special, and it will simplify the setup for you, but you will loose certain features such as the ability to list all the files saved during any particular job. We recommend against using the internal

database as it is being phased out. For more details please see the Internal Bacula Database chapter of this manual.
- At this point, you should have Bacula built and installed. If not, please follow the instructions in the Bacula Installation Chapter of this manual.

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**. For the details of doing the install, please see the Installing Bacula chapter of this manual.

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 vise versa, and **Bacula** will tell you want 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.

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.

Configuring the Console Program

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 **console.conf**. Normally, for first time users, no change is needed to this file. Reasonable defaults are set.

Configuring the File 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.

Configuring the Director

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 might use **foobaz−dir**, and for the storage daemon, you might use **foobaz−sd**.

Configuring the Storage 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**.

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
./console -t -c console.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.

Testing Bacula 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 btape −− Testing Your Tape Drive section of the Bacula Utility Programs chapter of this manual. If you have a modern standard SCSI tape drive, most likely it will work, but better test than be sorry.

Running Bacula

Please see the Running Bacula Chapter of this manual for detailed instructions on how to run **Bacula**

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 should see 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.


The Current State of Bacula

Index

Installing Bacula

Installing Bacula

General

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 **readline**, **gmp**, **cweb**, **SQLite**, **MySQL** to build properly depending on the options you specify. To simplify your task, we have combined a number of these packages into two **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.

Dependency Packages

As discussed above, we have combined a number of third party packages that **Bacula** might need into the **depkgs** and **depkgs1** releases. Typically, they will be named **depkgs−ddMMMyy.tar.gz** and **depkgs1−ddMMyy.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 **depkg** 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	depkgs1
SQLite	X	−
mtx	X	−
readline	−	X
gmp	−	X
cweb	−	X

Note, some of these packages are quite large, so that this part 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** and/or **depkgs1** prior to configuring and building **Bacula**, since **Bacula** will need them during the build process.

Even if you do not use SQLite, you might find it worth while 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, ...).

Building Bacula from Source

The basic installation is rather simple.

1. Install and build any **depkgs** as noted above.
2. Configure and install MySQL (if desired). Installing and Configuring MySQL Phase I. This is necessary so that the MySQL header files are available while compiling **Bacula**. Note, if you already have a running MySQL on your system, you can skip this phase.
3. As an alternative to MySQL, configure and install SQLite, which is part of the **depkgs**. Installing and Configuring SQLite.
4. Detar the **Bacula** source code preferably into the **bacula** directory discussed above.
5. **cd** to the directory containing the source code.
6. ./configure (with appropriate options as described below)
7. Check the output of ./configure very carefully, especially the Install binaries and Install config files 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**.
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.

9. make install
10. Create valid 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. Please take care when modifying passwords, which were randomly generated, and the daemon **Name**s as the passwords and names must agree between the configuration files for security reasons.
11. Create the Bacula MySQL databases and tables (if using MySQL) Installing and Configuring MySQL Phase II or alternatively if you are using SQLite Installing and Configuring SQLite Phase II.
12. Start Bacula (**./bacula start**)
13. Interface with Bacula using the Console program

If all goes well, the **./configure** will correctly determine which operating system you are running and configure the source code appropriately. Currently, FreeBSD, Linux (RedHat), and Solaris are supported.

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 RedHat, 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=$HOME/mysql \
    −−with−working−dir=$HOME/bacula/bin/working \
    −−with−dump−email=$USER \
    −−with−baseport=9101
```

For the developer's convenience, I have added a **defaultconfig** script to the **examples** directory. This script 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.

What Database to Use?

Before building **Bacula** you need to decide if you want to use SQLite, MySQL or the internal Bacula database. If you are not already running MySQL, we recommend that you start by using SQLite. This will greatly simplify the setup for you. You may also choose to use the internal database (recommended only for developers), as the ability to list all the files saved during any particular job. For serious backup, you will almost certainly want to use SQLite or MySQL.

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.

If you wish to use SQLite as the Bacula catalog, please see Installing and Configuring SQLite chapter of this manual.

If you wish to use the internal Bacula database, you need do nothing special, but for the features and limitations, please see Internal Bacula Database chapter of this manual.

Configure Options

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

−−sysbindir=<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.

−−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. This configuration parameter is used while building **Bacula**

−−enable−gnome
: If you have GNOME installed on your computer 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/gnome−console** directory.

−−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.

−−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.

−−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.

−−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.

−−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 build. 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.

−−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**.

−−with−sqlite=<sqlite−path>
This enables use of the SQLite 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 MySQL chapter of this manual for more details.

−−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. If this option is not present, the build will automatically include the internal **Bacula** database code. We recommend that you use this option if possible. If you do use this option, please proceed to installing MySQL in the Installing and Configuring MySQL chapter before proceeding with the configuration.

−−with−readline=<readline−path>
Tells Bacula where 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.

−−with−tcp−wrappers=<path>
This specifies that you want TCPWrappers compiled in (untested). The path is optional since Bacula will normally find the libraries in the standard locations. This option affects the **Bacula** build. **Please note that this option has not been fully tested.**

−−with−gmp=[DIR]
This option allows you to specify where **Bacula** can find the GNU Multi−precision library. Normally, if it is installed on your system, you will not need this option. This option affects the **Bacula** build.

−−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.

−−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**.

−−with−subsys−dir=<PATH>
This specifies where Bacula should place the subsystem lock file during execution. The default is **/var/run/subsys**.

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

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

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 \
  --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. See below for an example of how Kern does it.

RedHat

Using SQLite:

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

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
```

Solaris

```
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
```

Win32

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

Windows Systems with CYGWIN Installed

If you wish to build from the source, and if you have CYGWIN version 1.3.10 and GCC 2.95.3−5 installed, it is possible to build the Win32 version of **Bacula** on a Windows machine. Please don't try any other versions of CYGWIN as there were known problems. In addition, **Bacula** is designed to be installed on a non−CYGWIN system. If you do install it on a system with CYGWIN installed, you are on your own, and you must take special care to install **Bacula** in the main CYGWIN directory (normally c:\cygwin) rather than in the root (c:\).

To date, the Win32 version has only been build on a Win98 SR2 system and WinMe with the above CYGWIN environment and all the available CYGWIN tools loaded. In addition, the builds were done running under the **bash** shell. As time permits, we will experiment with other environments, and if any of you do build it from source, please let us know. The current CYGWIN environment was loaded using the CYGWIN setup.exe program, downloading ALL the latest binaries and installing them.

Note, although most parts of **Bacula** build on Windows systems, the only part that we have tested and used is the File daemon.

We recommend that you run the **./configure** command with the following options:

```
./configure \
  --sbindir=/bacula/bin \
  --sysconfdir=/bacula/bin \
  --with-working-dir=/bacula/working \
  --with-pid-dir=/bacula/working \
  --with-subsys-dir=/bacula/working \
  --enable-smartalloc \
  --with-baseport=9101
```

Note, the automatic installation for Win32 is not yet written, so most of these specifications are not really used.

After which, you can do a:

```
make
```

And to install you must do it by hand. During linking of some of the binaries, the following warning message is printed:

```
/USR/BIN/ld: warning: cannot find entry symbol _WinMainCRTStartup;
             defaulting to 00401000
```

This warning causes no harm. If there is some CYGWIN guru out there who knows how to eliminate this error, please let us know.

All the daemons will be built, but the only one tested to date is the Win32 File daemon. For the other daemons, you are on your own. It is not very probable they will work.

Finally, you should follow the installation instructions in the Win32 Installation section of this document, skipping the part that describes unZipping the binary release.

Kern's Configure Script

The script that I use for building on my "production" Linux machines is:

```
CFLAGS="-g -Wall" \
  ./configure \
    --sbindir=$HOME/bacula/bin \
    --sysconfdir=$HOME/bacula/bin \
    --enable-gnome \
    --with-pid-dir=$HOME/bacula/bin/working \
    --with-subsys-dir=$HOME/bacula/bin/working \
    --with-mysql=$HOME/mysql \
    --with-working-dir=$HOME/bacula/bin/working \
    --with-dump-email=$USER \
    --with-baseport=9101
exit 0
```

Note that I define the base port as 9101, which means that **Bacula** will use port 9101 for the Director console, port 9102 for the File daemons, and port 9103 for the Storage daemons. These ports should be available on all systems because they have been officially assigned to **Bacula** by IANA (Internet Assigned Numbers Authority). We strongly recommend that you use only these ports to prevent any conflicts with other programs. This is in fact the default if you do not specify a **--with-baseport** option.

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
```

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 O/S 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. This compiles with the built−in database and thus avoids the necessity of installing one or another of those database programs to build the File daemon. Finally, instead of using **make**, you can build just the File daemon (and the necessary libraries) with:

```
make bacula−fd
```

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. 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, RedHat Linux, and Solaris), and has only been fully tested on RedHat Linux.

The **make install−autostart** will cause the appropriate startup scripts to be installed with the necessary symbolic links. On RedHat 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
```

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 Kern uses as 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.

Modifying the Bacula Configuration Files

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

Customizing the Configuration Files

When each of the three **Bacula** daemons 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 Records. These resources are very similar from one service to another, but may contain different 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.

Resource Record Format

Although, you won't need to know the details of all the records, a basic knowledge of **Bacula** resource records is essential. Each record 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 record, and may optionally contain a Description record. The Name record is used to uniquely identify the resource. The Description record is (will be) used during display of the record to provide easier human recognition of the record. 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.

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.

Upper and Lower Case and Spaces

Case (upper/lower) and spaces are totally ignored in the resource record 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 slash (\) is taken as itself (handy for inserting slashes and double quotes (")). Please note, however, that **Bacula** resource names as well as certain other names (e.g. Volume names) will in the future be severely limited to permit only letters (including ISO accented letters), numbers, and a few special characters (space, underscore, ...). All other characters and punctuation will be illegal.

Recognized Primitive Data Types

When parsing the resource records, **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 straight forward.

name
: A keyword or name consisting of alpha numeric characters, including the hyphen, underscore, and dollar characters. (Note, currently other characters are not excluded). A name has a maximum length currently set to 127 bytes. Typically names appear on the left side of an equal (i.e. they are **Bacula** keywords). Names 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 virtually any character 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.

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)

m
: 1,048,576 (megabytes)

g
: 1,073,741,824 (gigabytes)

If you want to input one million rather than the megabyte size (1,048,576), you can simply enter **1.0e6**.

time

A time or durations specified in seconds. Typically, this is a floating point 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:

s
: seconds

n
: minutes (60 seconds)

h
: hours (3600 seconds)

d
: days (3600*24 seconds)

w
: weeks (3600*24*7 seconds)

m
: months (3600*24*30 seconds)

q
: quarters (3600*24*91 seconds)

y
: years (3600*24*365 seconds)

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 records from scratch.

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

The details of each Resource and the records 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.

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 file and resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual.

Director Resource Types

Director resource type may be one of the following:

Job, Client, Storage, Catalog, Schedule, FileSet, Pool, Director, or Messages.

We present them here in the most logical order for defining them:

- 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.
- Job −− to define the backup/restore Jobs and to tie together the Client, FileSet and Schedule resources to be used for each Job.
- Schedule −− to define when a Job is to be automatically run by **Bacula's** internal scheduler.
- FileSet −− to define the set of files to be backed up for each Client.
- Client −− to define what Client is to be backed up.
- Storage −− to define on what physical device the Volumes should be mounted.
- Pool −− to define what the pool of Volumes that can be used for a particular Job.
- Catalog −− to define in what database to keep the list of files and the Volume names where they are backed up.
- Messages −− to define where error and information messages are to be sent or logged.

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 records. One and only one director resource must be supplied.

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

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

Password = <UA−password>
: Specifies the password that must be supplied for a 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 actually passed across the network but rather a challenge response hash code created with the password. This record is required.

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 record 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. 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 record 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. 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. This record is required.

SubSys Directory = <Directory>
: This directive is mandatory and specifies a directory in which the Director may put its subsystem lock files. 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/subsys**. If you are not installing **Bacula** in the system directories, you can use the **Working Directory** as defined above. **Take care that you do not set this to the same directory that contains your binary files or they will be deleted.** This record is required.

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 record is required.

Maximum Concurrent Jobs = <number>

where <number> is the maximum number of Jobs that should run concurrently. The default is set to 1, but you may set it to a larger number. Note however, at this time (Bacula version 1.27), multiple simultaneous jobs have not been heavily tested. Because this feature is not yet well tested, we recommend that you either set it to 1 or make careful tests to ensure that everything you want works. The Volume format becomes much more complicated with multiple simultaneous jobs, and not all the utility programs (e.g. **bextract**, ... have been properly updated to deal with more than one Job at a time).**BE WARNED!!!!**

At the current time, there is no configuration parameter set or limit the number console connections. A maximum of five simultaneous console connections are permitted.

FD Connect Timeout = <time>

where **time** is the time in seconds 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 in seconds 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.

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 record need not be specified.

DirAddress = <IP−Address>

This record is optional, and 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 record is not specified, the Director will bind to any available address (the default).

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"
  SubSysDirectory = "$HOME/bacula/bin/working"
  QueryFile = "$HOME/bacula/bin/query.sql"
  Messages = Standard
}
```

The Job Resource

The Job resource defines a Job (Backup, Restore, ...) that Bacula must perform. Each Job resource definition contains the names of the Clients and their FileSets to backup or restore, 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.

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.

Job
: Start of the Job records. 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 record is required.

Type = <job−type>
: The **Type** record specifies the Job type, which may be one of the following: **Backup**, **Restore**, **Verify**, or **Admin**. This record is required.

  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 records are generated since no Files are saved.

  Verify
  : Run a verify Job. This is a very useful feature that allows you to use **Bacula's** database to ensure that critical system files are not modified by hackers. If the Verify job is level InitCatalog a record for each File will be placed in the catalog. However, for a simple level Catalog Verify job, only summary information on the job goes into the catalog.

  Admin
  : Run a 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 record specifies the default Job level to be run. The Level is normally overridden by a different value that is specified in the **Schedule** resource. This record is

not required, but must be specified either by a **Level** record or as a override specified in the **Schedule** resource.
For a **Backup** Job, the Level may be one of the following:

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

Incremental
is all files that have changed since the last successful backup of the specified FileSet.

Differential
is all files that have changed since the last successful Full backup of the specified FileSet.
For a **Restore** Job, no level need 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. At first glance, 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 this command one time when your system is first setup, and then once again after each modification (upgrade) to your system. Thereafter, you use a **Verify level = Catalog** to compare the results of your **InitCatalog** with the current state of the files.

Catalog
Compares the current state of the files against the value 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 the Verify Catalog modifies the Catalog database while running.

VolumeToCatalog
This level causes Bacula to read the data written to the Volume in the last Job. The data are compared to the values saved in the Catalog and any differences are reported. This is similar to the **Catalog** level except that instead of reading the data the disk, the data on the tape is read and compared.
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.

Bootstrap = <bootstrap−file>

The Bootstrap record 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 record is optional, and if specified, it is used only for a restore job. In addition, when running a Restore job from the console 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** record specifies a file name where **Bacula** will write a **bootstrap** file for each Backup job run. Thus this record 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 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.

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.

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

Client = <client−resource−name>
The Client record 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 record is required.

FileSet = <FileSet−resource−name>
The FileSet record 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 record is required.

Messages = <messages−resource−name>
The Messages record 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 record is required.

Pool = <pool−resource−name>
The Pool record 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

resource is required.

Schedule = <schedule−name>

The Schedule record defines what schedule is to be used for the Job. The schedule determines when the Job will be automatically started and what Job level (i.e. Full, Incremental, ...) is to be run. For additional details, see the Schedule Resource Chapter of this manual. If a Schedule resource is specified, the job will be run according to the schedule specified. If no Schedule resource is specified for the Job, the job must be manually started using the Console program. Although you may specify only a single Schedule resource for any one job, the Schedule resource may contain multiple **run** records, which allow you to run the Job at many different times, and each **run** record permits overriding the default Job Level Pool, Storage, and Messages resources. This gives considerable flexibility in what can be done with a single Job.

Storage = <storage−resource−name>

The Storage record 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. This record is required.

**Max Run Time = <seconds>*

The seconds specify the maximum runtime permitted for this Job. If the runtime exceeds this value, the job will be canceled. The default is 0 which indicates no maximum time limit. (NOT YET IMPLEMENTED).

Max Start Delay = <seconds>

The seconds specify 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.

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** record. If this record 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** record. If this record 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** record. If this record is specified (not normally) and the value is **yes**, it will override the value specified in the Client resource. The default is **no**.

Run Before Job = <command>

The specified **command** is run as an external program prior to running the current Job. This record is not required. The command string must be a valid program name or name of a shell 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
%i = JobId
%e = Job Exit
```

```
%j = Job
%l = Job Level
%n = Job name
%t = Job type
```

Run After Job = <command>
The specified **command** is run as an external program after the current job terminates. This record is not required. The command string must be a valid program name or name of a shell script. Before submitting the specified command to the operating system, **Bacula** performs character substitution as described above for the **Run Before Job** record.
An example of the use of this command is given in the Tips Chapter of this manual.

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. The default is **no**.

Where = <directory>
This record 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.

prefixlinks=<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** during restoration of 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. Although implemented in the Director, this status is not yet passed to the File daemon (for compatibility reasons). It will be fully enabled in version 1.30.

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
}
```

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. In general, you specify an action to be taken and when.

Schedule
: Start of the Schedule records. 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 record is required.

Run = <Job−overrides> <Date−time−specification>
: The Run record defines when a Job is to be run, and what overrides if any to apply. You may specify multiple **run** records within a **Schedule** resource. If you do, they will all be applied (i.e. multiple schedules). If you have two **run** records that start at the same time, two Jobs will start at the same time (well, within one second of time difference).
The **Job−overrides** permit overriding the **Level**, the **Storage**, the **Messages**, and the **Pool** specifications provided in the Job resource. By the use of these 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.

The **Job−overrides** are specified as: **keword=value** where the keword is **Level**, **Storage**, **Messages**, or **Pool**, and the **value** is as defined on the respective record formats for the Job resource. You may specify multiple **Job−overrides** on one **Run** record 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.

The **Date−time−specification** allows you to specify when the Job is to be run. Any specification given is assumed to be repetitive in nature. For example, **daily** means every day of every month in every year.

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.

The Job will be run on either day that matches the current day (day of the week, or day of the month).

The default is that every hour of every day of every week of every month is set. As you specify the parts of the time, the default for that part of the time is cleared and the new value set. However, the other defaults are set until their corresponding part is set. 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.

The following special keywords specify multiple parts of the time (e.g. **day** and **hour**), and in specifying them none of the other defaults are cleared:

```
Keyword       Meaning
===========   ================
Hourly        Every hour of every day of every month
Weekly        Every Sunday of the week of every month
Daily         Every day of every month
Monthly       Every first day of every month
```

All the other keywords show below specify only a single part of the time, and specifying them will clear all the defaults, which means that you must then specify all parts of the time:

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 | forth | fifth
<wday−keyword>    = sun | mon | tue | wed | thu | fri | sat |
                    sunday | monday | tuesday | wednesday |
                    thursday | friday
<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−range> | <wday−range> |
```

```
                        <daily-keyword>
      <day-spec>        = <day> | <wday-keyword> |
                        <week-keyword> <wday-keyword>
      <month-spec>      = <month-keyword> | <month-range> |
                        <monthly-keyword>
      <date-time-spec>  = <month-spec> <day-spec> <time-spec>
```

An example schedule resource that is named **WeeklyCycle** and runs a job with level full each Sunday at 1:05am and an incremental job Monday through Saturday at 1:05am is:

```
Schedule {
  Name = "WeeklyCycle"
  Run = Level=Full sun at 1:05
  Run = Level=Incremental mon-sat at 1:05
}
```

The FileSet Resource

The FileSet resource defines what files are to be included in a backup job. At least one **FileSet** resource is required. 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.

FileSet
: Start of the FileSet records. At least one **FileSet** resource must be defined.

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

Include = <processing−options>
{ <file−list> }
: The Include resource specifies the list of files and/or directories to be included in the backup job. There can be any number of **Include file−list** specifications within the FileSet, each having its own set of **processing−options**. Normally, the **file−list** consists of one file or directory name per line. Directory names should be specified without a trailing slash. Wild−card (or glob matching) can be specified. As a consequence, any asterisk (*), question mark (?), or left−bracket ([) must be preceded by a slash (\\) if you want it to represent the literal character.

  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**).

  Bacula's default for processing directories is to recursively descend in the directory saving all files and subdirectories. Bacula will not by default cross file systems (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 file systems. Similarly on Windows systems, you must explicitly specify each of the drives you want saved (e.g. **c:/** and **d:/** ...). The **df** command on Unix systems will show you which mount points you must specify to save everything. See below for an example.

  The **<processing−options>** is optional. If specified, it is a list of **keyword=value** options to be applied to the file−list. Multiple options may be specified by separating them with spaces. These options are used to modify the default processing behavior of the files included. Since there can be multiple **Include** sets, this permits effectively specifying the desired options (compression, encryption, ...) on a file by file basis. The options 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. However, 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. Specifying **GZIP** uses the default compression level six (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 6 generally give very little extra compression but are rather CPU intensive.

signature=MD5
: An MD5 signature will be computed for all files saved. Adding this option generates about 5% extra overhead for each file saved. We strongly recommend that this option be specified as a default for all files.

**encryption=<algorithm>*
: All files saved will be encrypted using one of the following algorithms (NOT YET IMPLEMENTED):
**Blowfish*
**3DES*

verify=<options>
: The options letters specified are used when running a **Verify Level=Catalog** job, and 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)
s
report file size decreases
5
compare the MD5 signature

A useful set of general options on the **Level=Catalog** 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. In this case, you must explicitly list each file system you want saved. If you set this 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 included in your FileSet, they will also be backed up. Normally, it

is preferable to set **onefs=yes** and to explicitly name each file system you want backed up. See the example below for more details.

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 contents of 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. The sparse code was first implemented in version 1.27.

**noexec=yes/no*

All executable files will be automatically excluded from being saved. The default is **no** i.e. all files including executable files are saved. (NOT YET IMPELEMENTED)

**nofollow=yes/no*

Any symbolic link found will be saved, but the link will not be followed (default). This is normally exactly what is desired because the file pointed to by the link will be also normally included in the save the same way the link was included. In some special cases, one might want to save just the contents of a directory and also include the contents of all links in that directory. This is possible by specifying **nofollow=no** (NOT YET IMPLEMENTED).

**nonfs=yes/no*

Any file that resides on an NFS file system will not be saved. The default is **no** so that any NFS file system specified in the **include** list will be backed up. (NOT YET IMPLEMENTED)

<file−list> is a space separated list of filenames and/or directory names. To include names containing spaces, enclose the name between double−quotes. The list may span multiple lines, in fact, normally it is good practice to specify each filename on a separate line.

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

- Any file−list item preceded by an at−sign (@) is assumed to be a filename containing a list of files, which is read when the configuration file is parsed during Director startup.

- Any file−list item beginning with a vertical bar (|) is assumed to be a program. This program will be executed by Bacula 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. 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.

  As an example:

```
Include = signature=MD5 {
   "|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 RedHat 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 preceeding 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 = signature=MD5 {
   "|my_partitions"
}
```

  where my_partitions has:

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

- Any file−list item preceded by a less−than sign (
- 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 = signature=MD5 sparse=yes {
   /dev/hd6
}
```

  will backup the data in device /dev/hd6.
- 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 = signature=MD5 readfifo=yes {
   /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, it will 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.

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.

The Exclude Files specifies the list of files and/or directories to be excluded from the backup job. The **<file−list>** is a comma separated list of filenames and/or directory names. To exclude names containing spaces, enclose the name between double−quotes. The list may span multiple lines. Any filename preceded by an at−sign (@) is assumed to be a filename containing a list of files.

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

```
FileSet {
  Name = "Full Set"
  Include = compression=GZIP signature=MD5 sparse=yes {
     @/etc/backup.list
  }
  Include = {
     /root/myfile
     /usr/lib/another_file
  }
  Exclude = { *.o }
}
```

Note, in the above example, all the files contained in /etc/backup.list will be compressed with GZIP compression, an MD5 signature will be computed on the file's contents (its data), and sparse file handling will apply.

The two files /root/myfile and /usr/lib/another_file will also be saved but without any options. In addition, all files with the extension **.o** will be excluded from the file set (i.e. from the backup).

Suppose you want to save everything except **/tmp** on your system. 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
```

Now, if you specify only / in your Include list, **Bacula** will only save the Filesystem **/dev/hda5**. To save all file systems except **/tmp** with out including any of the Samba or NFS mounted systems, you can use the following:

```
FileSet {
  Name = Everything
  Include = {
     /
     /boot
     /home
     /rescue
     /usr
  }
}
```

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 = onefs=no {
     /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. However, 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.

The following FileSet definition will backup a raw partition:

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

Note, in 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** record.

Windows Considerations for 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 = {
     "c:/My Documents"
  }
  Exclude = { *.obj *.exe }
}
```

Windows NTFS Naming Considerations

NTFS filenames containing Unicode characters (i.e. > 0xFF) cannot be explicitly named at the moment. You must include such names by naming a higher level directory or a drive letter that does not contain Unicode characters.

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 records.

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

File Retention = <time−period−specification>
The File Retention record defines the length of time that **Bacula** will keep File records in the Catalog database. 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. This period is specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of modifier specification.
The default is 60 days.

Job Retention = <time−period−specification>
The Job Retention record defines the length of time that **Bacula** will keep Job records in the Catalog database. 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. Note, 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 is specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of modifier 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.

**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
  Address = minimatou
  Catalog = MySQL
  Password = very_good
}
```

The Storage Resource

The Storage resource defines which Storage daemons are available for use by the Director.

Storage
: Start of the Storage records. At least one storage resource must be specified.

Name = <name>
: The name of the storage resource. This name appears on the Storage record specified in the Job record and is required.

Address = <address>
: Where the address is a host name, a **fully qualified domain name**, or a **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 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.

Device = <device−name>
: This record specifies the name of the device to be used to for the storage. This name is not the physical device name, but the logical device name as defined on the **Name** record contained in the **Device** 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**). This record is required.

Media Type = <MediaType>
: This record 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, ...). The **MediaType** specified here, **must** correspond to the **Media Type** specified in the **Device** resource of the **Storage daemon** configuration file. This record 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 record 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.

Auto Changer = <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** command in the console program. You may include this record whether the Storage device is really an autochanger or not. It will do no harm, but the Slot information will simply be ignored by the Storage daemon if the device is not really an autochanger. The default is **no**.

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

```
Storage {
  Name = DLTDrive
  Address = lpmatou
  Password = local_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
}
```

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 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.

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. However, if you change the Pool definition in the Director's configuration file and restart Bacula, then you will need to 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.

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 Volumes 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 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. (It really probably should do an **update pool** if you change the Pool definition, but currently, you must do this manually do so using the Console program.

The Pool Resource defined in the Director's configuration file (bacula−dir.conf) may contain the following records:

Pool
: Start of the Pool records. The 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 record is required.

Number of Volumes = <number>
: This record specifies the number of volumes (tapes or files) contained in the pool. Normally, it is defined and updated automatically by the **Bacula** catalog handling routines.

Maximum Volumes = <number>
: This record specifies the maximum number of volumes (tapes or files) contained in the pool. This record is optional, if omitted or set to zero, any number of volumes will be permitted. In general, this record is useful for Autochangers where there is a fixed number of Volumes, or for File storage where you wish to to ensure that the backups made to disk files do not become too numerous or consume too much space.

Pool Type = <type>
: This record 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**

Use Volume Once = <yes/no>
: This record 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 record will most likely be phased out (deprecated), so you are recommended to use **Maximum Volume Jobs = 1** instead.

Maximum Volume Jobs = <positive−integer>
: This record 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. By setting **MaximumVolumeJobs** to one, you get the same effect as setting **UseVolumeOnce = yes**.

Maximum Volume Files = <positive−integer>
: This record 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.

Maximum Volume Bytes = <size>
: This record 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.

Volume Use Duration = <time−period−specification>

The Volume Use Duration record 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, when 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.
You might use this record, 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.

Catalog Files = <yes/no>

This record 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).

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 record defines the length of time that **Bacula** will keep Job records associated with the Volume in the Catalog database. When this time period expires, and if **AutoPrune** is set to **yes Bacula** will prune (remove) Job records that are older than the specified Volume Retention period. This period is specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years.
The default is 365 days. Note, this record sets the default value for each Volume entry in the Catalog. The value in the catalog may be later individually changed for each Volume using the Console program.

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.

Recycle = <yes/no>

This record specifies the default for recycling Purged Volumes. If it is set to **yes** and **Bacula** needs a volume but finds none that are appendable, it will search for 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.

Accept Any Volume = <yes/no>

This record specifies whether or not any volume from the Pool may be used for backup. The default is **yes** as of version 1.27 and later. If it is **no** then only the first writable

volume in the Pool will be accepted for writing backup data, thus forcing Bacula to fill each Volume sequentially in turn before using any other appendable volume in the Pool. If this is **no** and you mount a volume out of order, **Bacula** will not accept it. If this is **yes** any appendable volume from the pool mounted will be accepted.
If you are going on vacation and you think the current volume may not have enough room on it, you can simply label a new tape and leave it in the drive, and assuming that **Accept Any Volume** is **yes** Bacula will begin writing on it. When you return from vacation, simply remount the last tape, and Bacula will continue writing on it until it is full. Then you can remount your vacation tape and Bacula will fill it in turn.

**Label Format = <format>*

This record specifies the format of the labels contained in this pool. The **format** consists of alpha−numeric characters and the special characters hyphen (−), underscore (_), colon (**:**), and period (**.**). In addition, there are special format commands that are enclosed in brackets that are of the form:

[<counter>:<specification

where **counter** must be a counter as specified on a **Counter** specification (see below), or a special built−in counter, and specification is either the name of a **FormatList** (see below), or one of the following format specifications:

"000", "AAA", "xxx"

with "xxx" denoting roman numerals.

The **:<specification>** is optional and if not given, the **counter** will be edited as a C printf **%d** value would be edited (i.e. a number of arbitrary width with no leading zero).

**Counter = <counter−name> [<initial−value>] [<maximum−value>] [<counter−to−increment>]*

where **counter−name** is the name of the counter to be used in a subsequent **Label Format** specification, and **initial−value** is an optional positive integer initial value for the counter, and **maximum−value** is an optional positive integer maximum value for the counter after which the counter will be reset to the initial value. The default **initial−value** is zero. The default and maximum **maximum−value** is 4,294,967,296. The optional name, **counter−to−increment** is the name of a counter to be incremented when the maximum value is exceeded (i.e. when the counter is reset to the initial value.

The counter is incremented when it is used (encountered in a **LabelFormat**) followed by a plus sign (e.g. [Counter+:000])

The following built−in counters exist:

Year
Month
Day
Hour
Minute
Second
Week

All counters including built−in counters have positive integer values.

**Format List = <list−name> = <arbitrary−list>*
where **arbitrary−list** is an arbitrary list of quoted strings of the form, "item1", "item2"item3", ...
The following built−in FormatLists exist:

UYear
Universal time and date
JDay
Julian day
Jtime
Julian day fraction
Shost
Storage daemon host name
Bhost
Backed up host name
Site
Site name
The counter value that precedes a built−in FormatList is ignored and may be omitted.

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 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
}
```

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, built−in) 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.

Catalog
: Start of the Catalog records. 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 record indicating that all catalog data for that Client is maintained in this Catalog. This record is required.

password = <password>
: This specifies the password to use when logging into the database. This record 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 record is required.

user = <user>
: This specifies what user name to use to log into the database. This record is required.

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

```
Catalog
{
  Name = SQLite
  dbname = bacula;
  user = bacula;
  password = ""                         # no password = no security
}
```

The Messages Resource

For the details of the Messages Resource, please see the Messages Resource Chapter of this manual.

A Complete Example Director Configuration File

An example Director configuration file might be the following:

```
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"
  SubSysDirectory = "/home/kern/bacula/bin/working"
  Password = "XkSfzu/Cf/wX4L8Zh4G4/yhCbpLcz3YVdmVoQvU3EyF/"
}

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
}


FileSet {
  Name = "Full Set"
  Include = signature=MD5 {

  /

  }
  Exclude = { }
}

Schedule {
  Name = "WeeklyCycle"
  Run = Full sun at 1:05
  Run = Incremental mon-sat at 1:05
}

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
}


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
}

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
}

Storage {
  Name = "8mmDrive"
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = "Exabyte 8mm"
  MediaType = "8mm"
}

Storage {
  Name = File
  Address = rufus
  Password = "jMeWZvfikUHvt3kzKVVPpQ0ccmV6emPnF2cPYFdhLApQ"
  Device = FileStorage
  Media Type = File
}


Catalog {
  Name = MyCatalog
  dbname = bacula; user = bacula; password = ""
}

Messages {
  Name = Standard
  mail = root@localhost = all, !skipped, !terminate
  operator = root@localhost = mount
  console = all, !skipped, !saved
}

Pool {
  Name = Default
  Pool Type = Backup
  AutoPrune = yes
  Recycle = yes
}
```



Bacula Configuration Index Client/File daemon Configuration

Bacula 1.29 User's Guide Chapter 5.2



Director Configuration Index Storage Daemon Configuration

Client/File daemon Configuration

General

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.

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 record 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. This record 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 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.

SubSys Directory = <Directory>
: This directive is mandatory and specifies a directory in which the Director may put its subsystem lock files. 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/subsys**. If you are not installing **Bacula** in the system directories, you can use the **Working Directory** as defined above. **Take care that you do not set this to the same directory that contains your binary files or they will be deleted.**

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.

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).

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
  SubSys Directory = $HOME/bacula/bin/working
}
```

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. This record 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 record is required.

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:

```
Director {
  Name = HeadMan
  Password = very_good                # password HeadMan must supply
}

Director {
  Name = Worker
  Password = not_as_good
}
```

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.

Example Client Configuration File

An example File Daemon configuration file might be the following:

```
Director {
  Name = rufus-dir
  Password = "/LqPRkX++saVyQE7w7mmiFg/qxYc1kufww6FEyY/47jU"
}

FileDaemon {                          # this is me
  Name = rufus-fd
  WorkingDirectory = $HOME/bacula/bin/working
  Pid Directory = $HOME/bacula/bin/working
  SubSys Directory = $HOME/bacula/bin/working
}

Messages {
  Name = Standard
  director = rufus-dir = all, !skipped
}
```


Director Configuration

Index

Storage Daemon Configuration


Client/File daemon Configuration

Index

Messages Resource

Storage Daemon Configuration

General

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 record types 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, very few records are actually needed.

Examples of **Device** resource records that are known to work for a number of common tape drives can be found in the **<bacula−src>/examples/devices** directory.

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.

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 record 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. This record 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 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.

SubSys Directory = <Directory>
: This directive is mandatory and specifies a directory in which the Director may put its subsystem lock files. 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/subsys**. If you are not installing **Bacula** in the system directories, you can use the **Working Directory** as defined above. **Take care that you do not set this to the same directory that contains your binary files or they will be deleted.**

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.

SDPort = <port−number>
Specifies port number on which the Storage daemon listens for Director connections. The default is 9103.

SDAddress = <IP−Address>
This record 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 record is not specified, the Storage daemon will bind to any available address (the default).

The following is a typical Storage daemon Storage definition.

```
Storage {
  Name = "Storage daemon"
  Address = localhost
  WorkingDirectory = "~/bacula/working"
  Pid    Directory = "~/bacula/working"
  SubSys Directory = "~/bacula/working"
}
```

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 record is required.

Password = <Director−password>
Specifies the password that must be supplied by the above named Director. This record is required.

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

```
Director {
  Name = MainDirector
  Password = my_secret_password
}
```

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** record described below.

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**". 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 modified the 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 a directory name. In that case, 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** record. 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 then **label** command to avoid attempting to write a label.

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 record is required.

Changer Device = name−string

The specified **name−string** gives the system file name of the autochanger device name that corresponds to the **Archive Device** specified. This device name is specified only if you have an autochanger. Typically, you will want to specify the **generic scsi** device name. For example, on Linux systems, for archive device **/dev/nst0**, This record is optional.

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 it to anything you want, but must be known to the volume database to keep track of which storage daemons can read which volumes. The same **name−string** must appear in the appropriate Storage resource definition in the Director's configuration file. This specification is required for all devices.

Auto Changer = Yes|No

If **Yes**, this device is an automatic tape changer, and you should also specify a **Changer Device** as well as a **Changer Command**. If **No** (default), the volume must be manually changed.

Changer Command = name−string

The **name−string** specifies an external program to be called that will automatically change volumes as required by **Bacula**. There are a number of substitution characters that may be specified to configure your autochanger. For additional details, please see the Autochangers chapter of this manual.

Maximum Changer Wait = time−in−seconds

This record specifies the maximum time 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 time out is 120 seconds.

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. 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. For File storage, this record is ignored. For a fifo storage device, you must set this to **No**.

Maximum Open Wait = time−in−seconds

This record 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).

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.

Maximum block size = size−in−bytes

The Storage daemon will aways 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).

Minimum block size = size−in−bytes

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 normally the case for 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. If you wish the block size to be fixed and different from the default, specify the same value for both **Minimum block size** and **Maximum block size**.

For example, suppose you want a fixed block size of 100K bytes, then you would specify:

```
Minimum block size = 100K
Maximum block size = 100K
```

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
```

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. Default 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.

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.

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 record 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. If you are using an autochanger, some devices require an offline to be issued prior to changing the volume.

Maximum Volume Size = size
No more than **size** bytes will be written onto a given volume on the archive device. This record 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 record 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.

Parallelism

Maximum Concurrent Jobs = positive integer
The storage daemon will accept no more than the given **positive integer** of simultaneous connections. The default is 2 so that you may have either two simultaneous jobs running or one job and one status connection. It is probably best to set this number fairly large (e.g. 10 or 20) and control how many Jobs are running with the **Maximum Concurrent Jobs** record in the Director's configuration file.

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 in each Pool resource. If this is **No** as by default, **Bacula** will label tapes only by specific operator command (**label**) or when the tape has been recycled.

Automatic mount = Yes|No
: If **Yes** (the default), permits the daemon to examine the device to determine if it contains the same **Bacula** labeled volume last recorded as mounted there in the daemon's internal status file. If so, re−mount in the same mode as it was mounted previously. While this slightly compromises security, it permits unattended servers to resume backup operations when the daemon is restarted on recovery from a system crash (or scheduled reboot).

Messages Resource

For a description of the Messages Resource, please see the Messages Resource Chapter of this manual.

Sample Storage Daemon Configuration File

A example Storage Daemon configuration file might be the following:

```
Storage {                             # definition of myself
  Name = rufus−sd
  Address = rufus
  WorkingDirectory = "$HOME/bacula/bin/working"
  Pid Directory = "$HOME/bacula/bin/working"
  Subsys Directory = "$HOME/bacula/bin/working"
}

Director {
  Name = rufus−dir
  Password = "ZF9Ctf5PQoWCPkmR3s4atCB0usUPg+vWWyIo2VS5ti6k"
}

Device {
  Name = "HP DLT 80"
  Media Type = DLT8000
  Archive Device = /dev/nst0
  AutomaticMount = yes;               # when device opened, read it
  AlwaysOpen = yes;
  RemovableMedia = yes;
}




Messages {
  Name = Standard
  director = rufus−dir = all
  operator = root = mount
```

```
}
```

  
Client/File daemon Configuration Index Messages Resource

  
Storage Daemon Configuration Index Console Configuration

Messages Resource

The 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, 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).

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**, ...), and **message−type** is one of a predefined set of keywords that define the type of message generated by **Bacula** (**ERROR**, **WARNING**, **FATAL**, ...).

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**, normally specified as a quoted string, the following substitutions will be used:
>
> - %% = %
> - %j = Job name

- %t = Job type (e.g. Backup, ...)
- %e = Job Exit code (OK, Error, ...)
- %l = Job level
- %c = Client's name
- %r = Recipients
- %d = Director's name

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/smtp −h mail.whitehouse.com −f \"Bacula \" −s \"Bacula: %t %e of %c %l\" %r"

Note, the **smtp** program is provided as part of **Bacula**. For additional details, please see the smtp −− Customizing Your Email Messages section of the Bacula Utility Programs chapter of this manual.

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**.

Debug = <debug−level>

This sets the debug message level to the debug level, which is an integer. Higher debug levels cause more debug information to be produced. You are requested not to use this record since it will be deprecated.

<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.

**broadcast*

broadcast the message to all users on the local machine except dialups similar to what **wall** does.

<destination> = <address> = <message−type1>, <message−type2...

Where **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_ERR** facility.

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 5 or 10 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.

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).

For any destination, the **message−type** field is a comma separated list of the following types of messages:

info
: information messages.

warning
: warning messages.

error
: non−fatal error messages. The job continues running.

fatal
: fatal error messages. Fatal errors cause the job to terminate.

terminate
: message generated when the daemon shuts down.

saved
: files saved normally.

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

all
: all message types.

**security*
: security info/warning messages (not currently implemented).

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.gov. In addition all mount messages are sent to the operator (i.e. emailed to enforcement@sec.gov). Finally all messages other than explicitly skipped files and files saved are sent to the console:

```
Messages {
  Name = Standard
  mail = enforcement@sec.gov = all, !skipped, !terminate
  operator = enforcement@sec.gov = mount
  console = all, !skipped, !saved
}
```

With the exception of the email address (changed to avoid junk mail from robot's), Kern's 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/smtp −h mail.whitehouse.com −f \"Bacula \"
                −s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "bacula/bin/smtp −h mail.whitehouse.com −f \"Bacula \"
                    −s \"Bacula: Intervention needed for %j\" %r"
  MailOnError = security@whitehouse.gov = all, !skipped, !terminate
  append = "bacula/bin/log" = all, !skipped, !terminate
  operator = security@whitehouse.gov = mount
  console = all, !skipped, !saved
}
```

  
Storage Daemon Configuration Index Console Configuration

Console Configuration

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 configuration file and resources including the data types recognized by **Bacula**, please see the Configuration chapter of this manual.

The following Console Resource definition must be defined:

- Director −− to define the Director's name and his access password. Note, you may define more than one Director resource in the Console configuration file. If you do so, the Console program will ask you which one you want to use.

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 records.

Name = <name>
: The director name used by the system administrator. This name must be identical to the **Name** specified in the **Director** resource of the Director's configuration file.

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.

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 record is required.

An actual example might be:

```
Director {
  Name = HeadMan
  address = rufus.sibbald.com
  password = xyz1erploit
}
```

Sample Console Configuration File

A example Console configuration file might be the following:

```
Director {
  Name = HeadMan
  address = "my_machine.my_domain.com"
  Password = Console_password
}
```


Messages Resource

Index

Running Bacula


Console Configuration

Index

The Console Program

Running Bacula

The general flow of running **Bacula** is:

1. cd <install−directory>
2. Start the Database (if using MySQL)
3. Start the Daemons with **./bacula start**
4. Start the Console program to interact with the Director
5. Run a job
6. When the tape fills, unmount the old tape, label a new one, and continue running
7. Test recovering some files from the tape just written to ensure the tape is good and that you know how to recover. Better test before disaster strikes

Each of these steps is described in more detail below.

Before Running Bacula

Before running **Bacula** for the first time, we recommend that you run the **test** command in the **btape** program as described in the Utility Program Chapter of this manual. This will help ensure that **Bacula** functions correctly with your tape drive. If you have a modern HP, Sony, or Quantum DDS or DLT tape drive running on Linux or Solaris, you can probably skip this test as **Bacula** is well tested with these drives and systems. For all other cases, you are **strongly** encouraged to run the test before continuing. **btape** also has a **fill** command that attempts to duplicate what Bacula does when filling a tape and writing on the next tape. You should consider trying this command as well, but be fore warned, it can take hours (about 4 hours on my drive) to fill a large capacity tape.

Starting the Database

If you are using MySQL as the **Bacula** database, you should start it before you attempt to run a job to avoid getting error messages from **Bacula** when it starts. The scripts **startmysql** and **stopmysql** are what I (Kern) use to start and stop my local MySQL. Note, if you are using the internal database or SQLite, you will not want to use **startmysql** or **stopmysql**. If you are running this in production, you will probably want to find some way to automatically start MySQL after each system reboot.

If you are using SQLite or the internal database (i.e. you did not specify the **−−with−mysql=xxx** option on the **./configure** command, you need do nothing. SQLite and the internal database are automatically started by **Bacula**.

Starting the Daemons

To start the three daemons, from the top level directory, simply enter:

./bacula start

This script starts the Storage daemon, the File daemon, and the Director daemon, which all normally run as daemons in the background.

Note, on Windows, currently only the File daemon is ported, and it must be started differently. Please see the Windows Version of Bacula Chapter of this manual.

The installation chapter of this manual explains how you can install scripts that will automatically restart the daemons when the system starts.

Daemon Command Line Options

Each of the three daemons (Director, File, Storage) accepts a small set of options on the command line. In general, each of the daemons as well as the Console program accepts the following options:

−c <file>
: Define the file to use as a configuration file. The default is the daemon name followed by **.conf** i.e. **bacula−dir.conf** for the Director, **bacula−fd.conf** for the File daemon, and **bacula−sd** for the Storage daemon.

−d nn
: Set the debug level to **nn**. Higher levels of debug cause more information to displayed on STDOUT concerning what the daemon is doing.

−f
: Run the daemon in the foreground. This option is needed to run the daemon under the debugger.

−s
: Do not trap signals. This option is needed to run the daemon under the debugger.

−t
: Read the configuration file and print any error messages, then immediately exit. Useful for syntax testing of new configuration files.

−?
: Print the version and list of options.

The Director has the following additional Director specific option:

−r <job>
: Run the named job immediately. This is for debugging and should not be used.

The File daemon has the following File daemon specific option:

−i
: Assume that the daemon is called from **inetd** or **xinetd**. In this case, the daemon assumes that a connection has already been made and that it is passed as STDIN. After the connection terminates the daemon will exit.

The Storage daemon has no Storage daemon specific options.

The Console program has no console specific options.

Interacting with the Director to Query or Start Jobs

To communicate with the director and to query the state of **Bacula** or run jobs, from the top level directory, simply enter:

./console

Alternatively, if you have GNOME installed and used the **−−enable−gnome** on the configure command, you may use the GNOME Console program:

./gnome−console

For simplicity, here we will describe only the **./console** program. Most of what is described here applies equally well to **./gnome−console**.

The **./console** runs the Bacula Console program, which connects to the Director daemon. Normally, it will print something similar to the following:

```
[kern@polymatou bin]$ ./console
Connecting to Director lpmatou:9101
1000 OK: HeadMan Version: 1.21 (20 June 2002)
*
```

the asterisk is the console command prompt.

Type **help** to see a list of available commands:

```
*help
  Command    Description
  =======    ===========
  add        add media to a pool
  autodisplay autodisplay [on/off] −− console messages
  automount  automount [on/off] −− after label
  cancel     cancel job=nnn −− cancel a job
  create     create DB Pool from resource
  delete     delete [pool= | media volume=]
  help       print this command
  label      label a tape
  list       list [pools | jobs | jobtotals | media  |
             files job=]; from catalog
  messages   messages
  mount      mount
  prune      prune expired records from catalog
  purge      purge records from catalog
  restore    restore files
  run        run
  setdebug   sets debug level
  show       show (resource records) [jobs | pools | ... | all]
  sqlquery   use SQL to query catalog
  status     status [storage | client]=
  unmount    unmount
  update     update Volume or Pool
  use        use catalog xxx
  version    print Director version
  quit       quit
  query      query catalog
  exit       exit = quit

*
```

Details of the console program's commands are explained in the Console Chapter of this manual.

Have Patience When Starting the Daemons or Mounting Blank Tapes

When you start the **Bacula** daemons, the Storage daemon attempts to open all defined storage devices and verify the currently mounted Volume (if configured). Until all the storage devices are verified, the Storage daemon will not accept connections from the Console program. If a tape was previously used, it will be rewound, and on some devices this can take several minutes. As a consequence, you may need to have a bit of patience when first contacting the Storage daemon after starting the daemons.

The same considerations apply if you have just mounted a blank tape in a drive such as an HP DLT. It can take a minute or two before the drive properly recognizes that the tape is blank. If you attempt to **mount** the tape with the Console program during this recognition period, it is quite possible that you will hang your SCSI driver (at least on my RedHat Linux system). As a consequence, you are again urged to have patience when inserting blank tapes. Let the device settle down before attempting to access it.

Last Steps Before Running a Job

There are only three more steps before you can run a job.

1. Create a Pool to hold Volume names. Bacula will automatically create all Pools you defined in the Director's configuration file.
2. You must label one or more Volume. That is physically write a label on the tape Volume. This can be done "on the fly" if you wish.
3. You must add one or more Volume names (tapes, or files) to the Pool database. If you use the **label** command, this last step will automatically be done.

Creating a Pool

If you understand Pools, you can skip to the next section.

When you run a job, one of the things that **Bacula** must know is what Volumes to use to backup the FileSet. Instead of specifying a Volume (tape) directly, you specify which Pool of Volumes you want **Bacula** to consult when it wants a tape for writing backups. Bacula will select the first available Volume from the Pool that is appropriate for the Storage device you have specified for the Job being run. When a volume is filled up with data, **Bacula** will change its Pool entry from **Append** to **Full**, and **Bacula** will use the next volume and so on. If no appendable Volume exists in the Pool, the Director will attempt to recycle an old Volume, if there are still no appendable Volumes available, **Bacula** send messages requesting the operator to create an appropriate Volume.

Bacula keeps track of the Pool name, the volumes contained in the Pool, and a number of attributes of each of those Volumes.

When **Bacula** starts, it ensures that all Pool resource definitions have been recorded in the catalog. You can verify this by entering:

```
list pools
```

to the console program, which should print something like the following:

```
*list pools
Using default Catalog name=MySQL DB=bacula
+--------+---------+---------+---------+----------+-------------+
| PoolId | Name    | NumVols | MaxVols | PoolType | LabelFormat |
+--------+---------+---------+---------+----------+-------------+
| 1      | Default | 3       | 0       | Backup   | *           |
| 2      | File    | 12      | 12      | Backup   | File        |
+--------+---------+---------+---------+----------+-------------+
*
```

If you attempt to create the same Pool name a second time, **Bacula** will print:

```
Error: Pool Default already exists.
```

Once created, you may use the **update** command to modify many of the values in the Pool record.

Labeling Your Volumes

Bacula requires that each Volume contain a software label. There are several strategies for labeling volumes. The one I use is to label them as they are needed by **Bacula** using the console program. That is when **Bacula** needs a new Volume, and it does not find one in the catalog, it will send me an email message requesting that I add Volumes to the Pool. I then use the **label** command in the Console program to label a new Volume and to define it in the Pool database, after which **Bacula** will begin writing on the new Volume.

Another strategy is to label a set of volumes at the start, then use them as **Bacula** requests them. This is most often done if you are cycling through a set of tapes. For more details on recycling, please see the Automatic Volume Recycling chapter of this manual.

If you run a **Bacula** job, and you have no labeled tapes in the Pool, **Bacula** will inform you, and you can create them "on−the−fly" so to speak. In my case, I label my tapes with the date, for example: **DLT−18April2002**. See below for the details of using the **label** command.

Labeling Volumes with the Console Program

Labeling volumes is normally done by using the console program.

1. ./console
2. label

If **Bacula** complains that you cannot label the tape because it is already labeled, simply **unmount** the tape using the **unmount** command in the console, then physically mount a blank tape and re−issue the **label** command.

Since the physical storage media is different for each device, the **label** command will provide you with a list of the defined Storage resources such as the following:

```
The defined Storage resources are:
     1: Floppy
```

```
     2: File
     3: 8mmDrive
     4: DLTDrive
     5: SDT-10000
Select Storage resource (1-5):
```

At this point, you should have a blank tape in the drive corresponding to the Storage resource that you select.

It will then ask you for the Volume name.

```
Enter new Volume name:
```

If **Bacula** complains:

```
Media record for Volume xxxx already exists.
```

It means that the volume name **xxxx** that you entered already exists in the Media database. You can list all the defined Media (Volumes) with the **list media** command. Note, the LastWritten column has been truncated for proper printing.

```
+---------------+-----------+-----------+----------------+-----/~/-+------------+---------
| VolumeName    | MediaType | VolStatus | VolBytes       | LastWri | VolReten   | Recycle
+---------------+-----------+-----------+----------------+---------+------------+---------
| DLTVol0002    | DLT8000   | Full      | 56,128,042,217 | 2001-10 | 31,536,000 |       0
| DLT-07Oct2001 | DLT8000   | Full      | 56,172,030,586 | 2001-11 | 31,536,000 |       0
| DLT-08Nov2001 | DLT8000   | Full      | 55,691,684,216 | 2001-12 | 31,536,000 |       0
| DLT-01Dec2001 | DLT8000   | Full      | 55,162,215,866 | 2001-12 | 31,536,000 |       0
| DLT-28Dec2001 | DLT8000   | Full      | 57,888,007,042 | 2002-01 | 31,536,000 |       0
| DLT-20Jan2002 | DLT8000   | Full      | 57,003,507,308 | 2002-02 | 31,536,000 |       0
| DLT-16Feb2002 | DLT8000   | Full      | 55,772,630,824 | 2002-03 | 31,536,000 |       0
| DLT-12Mar2002 | DLT8000   | Full      | 50,666,320,453 | 1970-01 | 31,536,000 |       0
| DLT-27Mar2002 | DLT8000   | Full      | 57,592,952,309 | 2002-04 | 31,536,000 |       0
| DLT-15Apr2002 | DLT8000   | Full      | 57,190,864,185 | 2002-05 | 31,536,000 |       0
| DLT-04May2002 | DLT8000   | Full      | 60,486,677,724 | 2002-05 | 31,536,000 |       0
| DLT-26May02   | DLT8000   | Append    |  1,336,699,620 | 2002-05 | 31,536,000 |       1
+---------------+-----------+-----------+----------------+-----/~/-+------------+---------
```

Once **Bacula** has verified that the volume does not already exist, it will then prompt you for the name of the Pool in which the Volume (tape) to be created. If there is only one Pool (Default), it will be automatically selected.

If the tape is successfully labeled, a media record will also be created in the Pool. That is the Volume name and all its other attributes will appear when you list the Pool. In addition, that Volume will be available for backup if the MediaType matches what is requested by the Storage daemon.

When you labeled the tape, you answered very few questions about it -- principally the Volume name, and perhaps the Slot. However, a Volume record in the catalog database (internally known as a Media record) contains quite a few attributes. Most of these attributes will be filled in from the default values that were defined in the Pool (i.e. the Pool holds most of the default attributes used when creating a Volume).

It is also possible to add media to the pool without physically labeling the Volumes. This can be done with the **add** command. For more information, please see the Console Chapter of this

manual.

Running a Job

If you have specified all the necessary Job resources including a schedule, your Backup jobs will be automatically scheduled. To find out what jobs are running and what jobs are scheduled to be run in the next 24 hours, enter the **status dir** command to the console program:

```
[kern@polymatou bin]$ ./console
Connecting to Director localhost:9101
1000 OK: rufus-dir Version: 1.15 (5 March 2002)
*status dir
HeadMan Version: 1.15 (5 March 2002)
Daemon started 05-Mar-2002 19:15, 0 Jobs run.
Console connected at 05-Mar-2002 19:15
No jobs are running.
Backup job "Rufus" scheduled for 06-Mar-2002 01:05
Backup job "Minou" scheduled for 06-Mar-2002 01:05
Backup job "Minimatou" scheduled for 06-Mar-2002 01:05
Verify job "MatouVerify" scheduled for 06-Mar-2002 05:05
Backup job "Matou" scheduled for 06-Mar-2002 01:05
Backup job "Polymatou" scheduled for 06-Mar-2002 01:05
====
*
```

In this case, you can see that no jobs are running, but that there are five jobs scheduled to be run at 1:05am (note they will run one at a time since the Director is configured for one job maximum).

Now to run a Job immediately, you can simply enter:

```
run Rufus
```

where **Rufus** is the name of the job to be run. The default job level will be used (normally incremental).

Alternatively, you can simply enter:

```
run
```

and **Bacula** will prompt you to choose a job to run from the list of all jobs.

Before actually running the job, **Bacula** will display the parameters of the job to be run and request you to confirm. In the following example, the user has requested to run job **kernsave**:

```
Run Backup job
JobName:  kernsave
FileSet:  Kerns Files
Level:    incremental
Client:   Rufus
Storage:  SDT-10000
OK to run? (yes/mod/no):
```

If you wish to run the job, simply enter **yes**. If you want to change the parameters shown before

running the job, enter **mod** for modify, and **Bacula** will prompt you for the values you wish to change.

You can use the **status** command to watch the progress of your job.

When the job completes, it will normally email you a message and send it to the Console (depending on your configuration). The output from a typical backup Job such as the one started above looks like the following:

```
rufus-dir: 19-Sep-2002 10:52
JobId:                  6
Job:                    kernsave.2002-09-19.10:50:48
FileSet:                Kerns Files
Backup Level:           Full
Client:                 Rufus
Start time:             19-Sep-2002 10:50
End time:               19-Sep-2002 10:52
Files Written:          85
Bytes Written:          4,181,712
Rate:                   50.4 KB/s
Volume names(s):        test5
Volume Session Id:      1
Volume Session Time:    1032425434
Volume Bytes:           25,158,136
Termination:            Backup OK

rufus-dir: Begin pruning Jobs.
rufus-dir: No Jobs found to prune.
rufus-dir: Begin pruning Files.
rufus-dir: No Files found to prune.
rufus-dir: End auto prune.
```

If you have defined a **Schedule** resource for your job, it will be run automatically according to the defined schedule.

Quitting the Console Program

Simply enter the command **quit**.

When The Tape Fills

If you have scheduled your job, typically nightly, there will come a time when the tape fills up and **Bacula** cannot continue. In this case, **Bacula** will send you a message similar to the following:

```
rufus-sd: block.c:337 === Write error errno=28: ERR=No space left
          on device
```

This indicates that **Bacula** got a write error because the tape is full. **Bacula** will then search the Pool specified for your Job looking for an appendable volume. In the simplest case, it will not find one, and it will send you a message similar to the following:

```
rufus-sd: Job kernsave.2002-09-19.10:50:48 waiting. Cannot find any
          appendable volumes.
```

```
Please use the "label"  command to create a new Volume for:
    Storage:      SDT-10000
    Media type:   DDS-4
    Pool:         Default
```

Until you create a new Volume, this message will be repeated an hour later, then two hours later, and so on doubling the interval each time up to a maximum interval of 1 day.

The obvious question at this point is: What do I do now?

The answer is simple: first, using the Console program, release the tape using the **unmount** command. If you only have a single drive, it will be automatically selected, otherwise, make sure you release the one specified on the message (in this case **STD-10000**).

Next, you remove the tape from the drive and insert a new blank tape. Note, on some older tape drives, you may need to write an end of file mark (**mt -f /dev/nst0 weof**) to prevent the drive from running away when **Bacula** attempts to read the label.

Finally, you use the **label** command in the Console to write a label to the new Volume. The **label** command will contact the Storage daemon to write the software label, if it is successful, it will add the new Volume to the Pool, then issue a **mount** command to the Storage daemon. See the previous sections of this chapter for more details on labeling tapes.

The result is that **Bacula** will continue the previous Job writing the backup to the new Volume.

Other Useful Console Commands

status dir
: Print a status of all running jobs and jobs scheduled in the next 24 hours.

status
: The console program will prompt you to select a daemon type, then will request the daemon's status.

status jobid=nn
: Print a status of JobId nn if it is running. The Storage daemon is contacted and requested to print a current status of the job as well.

list pools
: List the pools defined in the Catalog (normally only Default is used).

list media
: Lists all the media defined in the Catalog.

list jobs
: Lists all jobs in the Catalog that have run.

list jobid=nn
: Lists JobId nn from the Catalog.

list jobtotals
: Lists totals for all jobs in the Catalog.

list files jobid=nn
: List the files that were saved for JobId nn.

list jobmedia
: List the media information for each Job run.

messages
: Prints any messages that have been directed to the console.

unmount storage=storage−name
: Unmounts the drive associated with the storage device with the name **storage−name** if the drive is not currently being used. This command is used if you wish **Bacula** to free the drive so that you can use it to label a tape.

mount storage=storage−name
: Causes the drive associated with the storage device to be mounted again. When **Bacula** reaches the end of a volume and requests you to mount a new volume, you must issue this command after you have placed the new volume in the drive. In effect, it is the signal needed by **Bacula** to know to start reading or writing the new volume.

quit
: Exit or quit the console program.

Most of the commands given above, with the exception of **list**, will prompt you for the necessary arguments if you simply enter the command name.

Debug Daemon Output

If you want debug output from the daemons as they are running, start the daemons from the install directory as follows:

```
./bacula start −d20
```

To stop the three daemons, enter the following from the install directory:

```
./bacula stop
```

The execution of **bacula stop** may complain about pids not found. This is OK, especially if one of the daemons has died, which is very rare.

To do a full system save, each File daemon must be running as root so that it will have permission to access all the files. None of the other daemons require root privileges. However, the Storage daemon must be able to open the tape drives. On many systems, only root can access the tape drives. Either run the Storage daemon as root, or change the permissions on the tape devices to permit non−root access. MySQL can be installed and run with any userid; root privilege is not necessary.


Console Configuration

Index

The Console Program

Bacula Console

General

The **Bacula Console** (sometimes called the User Agent) is a program that allows the user or the System Administrator, to interact with the Bacula Director daemon while the daemon is running.

The current Bacula Console comes in two versions: a shell interface (TTY style), and a GNOME GUI interface. Both permit the administrator or authorized users to interact with Bacula. You can determine the status of a particular job, examine the contents of the Catalog as well as perform certain tape manipulations with the Console program.

Since the Console program interacts with the Director by the network, your Console and Director programs do not necessarily need to run on the same machine.

In fact, a certain minimal knowledge of the Console program is needed in order for Bacula to be able to write on more than one tape, because when Bacula requests a new tape, it waits until the user, via the Console program, indicates that the new tape is mounted.

Configuration

When the Console starts, it reads a standard Bacula configuration file named **console.conf** or **gnome−console.conf** in the case of the GNOME Console version. This file allows default configuration of the Console, and at the current time, the only Resource Record defined is the Director resource, which gives the Console the name and address of the Director. For more information on configuration of the Console program, please see the Console Configuration File Chapter of this document.

Running the Console Program

After launching the Console program, it will prompt you for the next command with an asterisk (*). (Note, in the GNOME version, the prompt is not present; you simply enter the commands you want in the command text box at the bottom of the screen.) Generally, for all commands, you can simply enter the command name and the Console program will prompt you for the necessary arguments. Alternatively, in most cases, you may enter the command followed by arguments. The general format is:

```
<command> <keyword1>[=<argument1>] <keyword2>[=<argument2>] ...
```

where **command** is one of the commands listed below; **keyword** is one of the keywords listed below (usually followed by an argument); and **argument** is the value.

For example:

```
list files jobid=23
```

will list all files saved for JobId 23. Or:

```
show pools
```

will display all the Pool resource records.

Stopping the Console Program

Normally, you simply enter **quit** or **exit** and the Console program will terminate. However, it waits until the Director acknowledges the command. If the Director is already doing a lengthy command (e.g. prune), it may take some time. If you want to immediately terminate the Console program, enter the **.quit** command.

There is currently no way to interrupt a Console command once issued (i.e. ctl−C does not work). However, if you are at a prompt that is asking you to select one of several possibilities and you would like to abort the command, you can enter a period (**.**), and in most cases, you will either be returned to the main command prompt or if appropriate the previous prompt (in the case of nested prompts).

Console Commands

The following commands are currently implemented:

add
: This command is used to add Volumes to an existing Pool. The Volume names entered are placed in the Catalog and thus become available for backup operations. Normally, the **label** command is used rather than this command because the **label** command labels the physical media (tape) and does the equivalent of the **add** command. This command affects only the Catalog and not the physical media. The physical media must exist and be labeled before use (usually with the **label** command). This command can, however, be useful if you wish to add a number of Volumes to the Pool that will be physically labeled at a later time. It can also be useful if you are importing a tape from another site.

autodisplay
: This command accepts **on** or **off** as an argument, and turns auto−display of messages on or off respectively. The default for the console program is **off**, which means that you will be notified when there are console messages pending, but they will not automatically be displayed. The default for the gnome−console program is **on**, which means that messages will be displayed when they are received (usually within 5 seconds of them being generated).
When autodisplay is turned off, you must explicitly retrieve the messages with the **messages** command. When autodisplay is turned on, the messages will be displayed on the console as they are received.

automount
: This command accepts **on** or **off** as the argument, and turns auto−mounting of the tape after a **label** command on or off respectively. The default is **on**. If **automount** is turned off, you must explicitly **mount** the tape after a label command to use it.

cancel
: This command is used to cancel a job and accepts **jobid=nnn** or **job=xxx** as an argument where nnn is replaced by the JobId and xxx is replaced by the job name. If you do not specify a keyword, the Console program will prompt you with the names of all the active jobs allowing you to choose one.
Once a Job is marked to be canceled, it may take a bit of time (generally within a minute) before it actually terminates, depending on what operations it is doing. If a Job is

waiting for a tape to be mounted, it will normally check every 20 minutes. You can cause it to be immediately check (and thus canceled) by issuing a **mount** command from the Console.

create

This command is used to create a Pool record in the database using the Pool resource record defined in the Director's configuration file. So in a sense, this command simply transfers the information from the Pool resource in the configuration file into the Catalog. Normally this is the very first command that you will issue when first communicating with the Director. It is typically done only once to create the Pool, and usually the first Pool (and possibly the only Pool) record that you will create will be for the **Default** Pool. Immediately after creating the Pool, you will most likely use the **label** command to label one or more volumes and add their names to the Media database. When starting a Job, when **Bacula** determines that there is no Pool record in the database, but there is a Pool resource of the appropriate name, it will create it for you.

setdebug

This command is used to set the debug level in each daemon. The form of this command is:
setdebug level=nn [client=<client−name> | dir | director | storage=<storage−name> | all]

delete

The delete command is used to delete a Pool record from the Catalog as well as all associated Volume records that were created. This command is very dangerous and we strongly recommend that you do not use it unless you know what you are doing.
The full form of this command is:

delete pool=<pool−name>

or

delete media pool=<pool−name> volume=<volume−name>

The first form deletes a Pool record from the catalog database. The second form deletes a Volume record from the specified pool in the catalog database.

help

This command displays the list of commands available.

label

This command is used to label physical volumes. The full form of this command is:
label storage=<storage−name> volume=<volume−name>

The media type is automatically taken from the Storage resource definition that you supply. Once the necessary information is obtained, the Console program contacts the specified Storage daemon and requests that the tape be labeled. If the tape labeling is successful, the Console program will create a Volume record in the appropriate Pool.

The label command can fail for a number of reasons:

1. The Volume name you specify is already in the Volume database.

2. The Storage daemon has a tape already mounted on the device, in which case you must **unmount** the device, insert a blank tape, then do the **label** command.
3. The tape in the device is already a Bacula labeled tape. (Bacula will never relabel a Bacula labeled tape).
4. There is no tape in the drive.

list

The list command lists particular contents of the Catalog. The various forms of the list command are:
list jobs

list jobid=<id>

list job=<job−name>

list jobmedia [jobid=<id> | job=<job−name>]

list files [jobid=<id> | job=<job−name>]

list pools

list jobtotals

list media

If you wish to add specialized commands that list the contents of the catalog, you can do so by adding them to the **query.sql** file. However, this takes some knowledge of programming SQL. Please see the **query** command below for additional information.

messages

This command causes any pending console messages to be immediately displayed.

mount

The mount command is used to mount a volume on a physical device. This command is used only if there is no Volume in a drive and **Bacula** requests a mount or when you have specifically unmounted a Volume with the **unmount** console command. The various forms of the mount command are:
mount storage=<storage−name>

mount [jobid=<id> | job=<job−name>]

If you have specified **Automatic Mount = yes** in the Storage daemon's Device resource, under most circumstances, **Bacula** will automatically access the Volume unless you have explicitly **unmount**ed it in the Console program.

prune

The Prune command allows you to safely remove associated records from Jobs and Volumes. In all cases, the Prune command applies a retention period to the specified records. You can Prune expired File entries from Job records; you can Prune expired Job records from the database, and you can Prune both expired Job and File records from specified Volumes.

purge

The Purge command will delete associated records from Jobs and Volumes without considering the retention period. This command is dangerous and we recommend that you do not use it unless you know what you are doing.

restore

The restore command allows you to select one or more Jobs (JobIds) to be restored using various methods. Once the JobIds are selected, the File records for those Jobs are placed in an internal **Bacula** directory tree, and the restore enters a file selection mode that allows you to interactively walk up and down the file tree selecting individual files to be restored. This mode is somewhat similar to the standard Unix **restore** program's interactive file selection mode.
For details of the **restore** command, please see the Restore Chapter of this manual.

run

This command allows you to schedule jobs to be run immediately. The full form of the command is:
run job=<job−name> client=<client−name> fileset=<FileSet−name>
level=<level−keyword> storage=<storage−name> where=<directory−prefix>

Any information that is needed but not specified will be listed for selection, and before starting the job, you will be prompted to accept, reject, or modify the parameters of the job to be run.

On my system, when I enter a run command, I get the following prompt:

```
A job name must be specified.
The defined Job resources are:
     1: Matou
     2: Polymatou
     3: Rufus
     4: Minimatou
     5: Minou
     6: PmatouVerify
     7: MatouVerify
     8: RufusVerify
     9: Watchdog
Select Job resource (1−9):
```

If I then select number 5, I am prompted with:

```
Run Backup job
JobName:  Minou
FileSet:  Minou Full Set
Level:    Incremental
Client:   Minou
Storage:  DLTDrive
OK to run? (yes/mod/no):
```

If I now enter **yes**, the Job will be run. If I enter **mod**, I will be presented with the following prompt.

```
Parameters to modify:
     1: Level
     2: Storage
```

```
     3: Job
     4: FileSet
     5: Client
Select parameter to modify (1-5):
```

show

The show command will list the Director's resource records as defined in the Director's configuration file (normally **bacula-dir.conf**). This command is used mainly for debugging purposes by developers.

sqlquery

The sqlquery command puts the Console program into SQL query mode where each line you enter is concatenated to the previous line until a semicolon (;) is seen. The semicolon terminates the command, which is then passed directly to the SQL database engine. When the output from the SQL engine is displayed, the formation of a new SQL command begins. To terminate SQL query mode and return to the Console command prompt, you enter a period (.) in column 1.
Using this command, you can query the SQL catalog database directly. Note you should really know what you are doing otherwise you could damage the catalog database. See the **query** command below for simpler and safer way of entering SQL queries.

Depending on what database engine you are using (MySQL or SQLite), you will have somewhat different SQL commands available. For more detailed information, please refer to the MySQL or SQLite documentation.

status

This command will display the status of the next jobs that are scheduled during the next twenty-four hours as well as the status of currently running jobs.

unmount

This command causes the indicated Bacula Storage daemon to unmount the specified device. The forms of the command are the same as the mount command:
unmount storage=<storage-name>

unmount [jobid=<id> | job=<job-name>]

update

This command will update either a specific Pool record or a Volume record in the Catalog. In the case of updating a Pool record, the new information will be pulled from the corresponding Director's configuration resource record. It can be used to increase the maximum number of volumes permitted or to set a maximum number of volumes.
In the case of updating a Volume, you will be prompted for which value you wish to change. The following Volume parameters may be changed:

```
Volume Status
Volume Retention Period
Recycle Flag
Slot
```

use

This command allows you to specify which Catalog database to use. Normally, you will be using only one database so this will be done automatically. In the case that you are using more than one database, you can use this command to switch from one to another.

version

The command prints the Director's version.

quit

This command terminates the console program. The console program sends the **quit** request to the Director and waits for acknowledgment. If the Director is busy doing a previous command for you that has not terminated, it may take some time. You may quit immediately by issuing the **.quit** command (i.e. quit preceded by a period).

query

This command reads a predefined SQL query from the query file (the name and location of the query file is defined with the QueryFile resource record in the Director's configuration file). You are prompted to select a query from the file, and possibly enter one or more parameters, then the command is submitted to the Catalog database SQL engine.
The following queries are currently available (version 1.24):

```
Available queries:
     1: List Job totals:
     2: List where a file is saved:
     3: List where the most recent copies of a file are saved:
     4: List total files/bytes by Job:
     5: List total files/bytes by Volume:
     6: List last 20 Full Backups for a Client:
     7: List Volumes used by selected JobId:
     8: List Volumes to Restore All Files:
     9: List where a File is saved:
Choose a query (1-9):
```

exit

This command terminates the console program.

Adding Media to a Pool

If you have used the **label** command to label a Volume, it will be automatically added to the Pool, and you will not need to add any media to the pool.

Alternatively, you may choose to add a number of Volumes to the pool without labeling them. At a later time when the Volume is requested by **Bacula** you will need to label it.

Before adding media, you must know the following information:

1. The name of the Pool (normally "Default")
2. The Media Type as specified in the Storage Resource in the Director's configuration file (e.g. "DLT8000")
3. The number and names of the Volumes you wish to create.

For example, to add media to a Pool, you would issue the following commands to the console program:

```
*add
Enter name of Pool to add Volumes to: Default
Enter the Media Type: DLT8000
Enter number of Media volumes to create. Max=1000: 10
```

```
Enter base volume name: Save
Enter the starting number: 1
10 Volumes created in pool Default
*
```

To see what you have added, enter:

```
*list media pool=Default
+---------+-----------+-----------+-----------+---------+---------------------+
| MediaId | VolumeName | MediaType | VolStatus | VolBytes | LastWritten         |
+---------+-----------+-----------+-----------+---------+---------------------+
|      11 | Save0001   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      12 | Save0002   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      13 | Save0003   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      14 | Save0004   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      15 | Save0005   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      16 | Save0006   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      17 | Save0007   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      18 | Save0008   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      19 | Save0009   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
|      20 | Save0010   | DLT8000   | Append    |        0 | 0000-00-00 00:00:00 |
+---------+-----------+-----------+-----------+---------+---------------------+
*
```

Notice that the console program automatically appended a number to the base Volume name that you specify (Save in this case). If you don't want it to append a number, you can simply answer 0 (zero) to the question "Enter number of Media volumes to create. Max=1000:", and in this case, it will create a single Volume with the exact name you specify.

Running Bacula Index The Console Restore Command

Catalog Maintenance

Without proper setup and maintenance, your Catalog may continue to grow indefinitely as you run Jobs and backup Files. How fast the size of your Catalog grows depends on the number of Jobs you run and how many files they backup. By deleting records within the database, you can make space available for the new records that will be added during the next Job. By constantly deleting old expired records (dates older than the Retention period), your database size will remain constant.

If you started with the default configuration files, they are already set with reasonable defaults for a small number of machines (less that 5), so if you fall into that case, catalog maintenance will not be urgent if you have a few hundred megabytes of disk space free. What ever the case may be, some knowledge of retention periods will be useful.

Retention Period

Bacula uses three Retention periods: the **File Retention** period, the **Job Retention** period, and the **Volume Retention** period. Of these three, the File Retention period is by far the most important in determining how large your database will become.

The **File Retention** and the **Job Retention** are specified in each Client resource as is shown below. The **Volume Retention** period is specified in the Pool resource, and the details are given in the next chapter of this manual.

File Retention = <time−period−specification>

The File Retention record defines the length of time that **Bacula** will keep File records in the Catalog database. 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. The pruning will occur at the end of a backup Job for the given Client. Note that the Client database record contains a copy of the File and Job retention periods, but **Bacula** uses the current values found in the Director's Client resource to do the pruning. Retention periods are specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years on the record. See the Configuration chapter of this manual for additional details of modifier specification.

The default is 60 days.

Job Retention = <time−period−specification>

The Job Retention record defines the length of time that **Bacula** will keep Job records in the Catalog database. 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. Note, 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 retention period is specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years. See the Configuration chapter of this manual for additional details of modifier 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 turn this off by setting it to **no**, your Catalog will grow each time you run a Job.

Compressing Your MySQL Database

Over time, as noted above, your database will tend to grow. I've noticed that even though **Bacula** regularly prunes files, **MySQL** does not effectively use the space, and instead continues growing. To avoid this, from time to time, you must compact your database. Normally, large commercial database such as Oracle have commands that will compact a database to reclaim wasted file space. However, neither MySQL nor SQLite have such a command. As a consequence, you must write the database out into an ASCII format, then reload it, effectively starting from scratch producing a compacted result. For a **MySQL** database, you could do it as follows:

```
mysqldump -f --opt bacula > bacula.sql
mysql bacula < bacula.sql
rm -f bacula.sql
```

There is no need to explicitly delete the old database as MySQL will automatically recreate the database. Depending on the size of your database, this will take more or less time and a fair amount of disk space. For example, if I cd to the location of the MySQL Bacula database (typically /opt/mysql/var or something similar) and enter:

```
du bacula
```

I get **620,644** which means there are that many blocks containing 1024 bytes each or approximately 635 MB of data. After doing the **msqldump**, I had a file that had **174,356** blocks, and after doing the **mysql** command to recreate the database, I ended up with a total of **210,464** blocks rather than the original **629,644**. In other words, the compressed version of the database took approximately one third of the space of the database that had been in use for about a year.

As a consequence, I suggest you monitor the size of your database and from time to time (once every 6 months or year), compress it.

Compressing Your SQLite Database

First please read the previous section that explains why it is necessary to compress a database. For SQLite, you can use the following commands, adapted to your system:

```
cd working-directory
echo '.dump' | sqlite bacula.db > bacula.sql
rm -f bacula.db
sqlite bacula.db < bacula.sql
rm -f bacula.sql
```

Where **working-directory** is the directory that you specified in the Director's configuration file. Note, in the case of SQLite, it is necessary to completely delete (rm) the old database before

creating a new compressed version.

Backing Up Your Bacula Database

If ever the machine on which you Bacula database crashes, and you need to restore from backup tapes, one of your first priorities will probably be to recover the database. Although **Bacula** will happily backup your catalog database if it is specified in the FileSet, this is not a very good way to do it because the database will be saved while Bacula is modifing it. Thus the database may be in and instable state. Worse yet, you will backup the database before all the Bacula updates have been applied.

To resolve these problems, you need backup the database after all the backup jobs have been run. In addition, you will want to make a copy while Bacula is not modifying it. To do so, you can use two scripts provided in the release **make_catalog_backup** and **delete_catalog_backup**. These files will be automatically generated along with all the other Bacula scripts. The first script will make an ASCII copy of your Bacula database into **bacula.sql** in the working directory you specified on your configuration, and the second will delete the **bacula.sql** file.

The basic squence of events to make this work correctly is as follows:

- Run all your nightly backups
- After running your nightly backups, run a Catalog backup Job
- The Catalog backup job must be scheduled after your last nightly backup
- You use **RunBeforeJob** to create the ASCII backup file and **RunAfterJob** to clean up

Assuming that you start all you nightly backup jobs at 1:05 am (and that they run one after another), you can do the catalog backup with the following additional Director configuration statements:

```
Job {
  Name = "BackupCatalog"
  Type = Backup
  Client=rufus-fd
  FileSet="Catalog"
  Schedule = "WeeklyCycleAfterBackup"
  Storage = DLTDrive
  Messages = Standard
  Pool = Default
  RunBeforeJob = "/home/kern/bacula/bin/make_catalog_backup"
  RunAfterJob  = "/home/kern/bacula/bin/delete_catalog_backup"
}

Schedule {
  Name = "WeeklyCycleAfterBackup
  Run = Full sun-sat at 1:10
}

FileSet {
  Name = "Catalog"
  Include = signature=MD5 {
     @working_directory@/bacula.sql
  }
}
```

Backing Up System Databases

If you are running a database in production mode on your machine, **Bacula** will happily backup the files, but if the database is in use while **Bacula** is reading it, you may back it up in an unstable state.

The best solution is to shutdown your database before backing it up, or use some tool specific to your database to make a valid live copy perhaps by dumping the database in ASCII format. I am not a database expert, so I cannot provide you advice on how to do this, but if you are unsure about how to backup your database, you might try visiting the Backup Central site, which has been renamed Storage Mountain (www.backupcentral.com). In particular, their Free Backup and Recovery Software page has links to scripts that show you how to shutdown and backup most major databases.

Database Size

As mentioned above, if you do not do automatic pruning, your Catalog will grow each time you run a Job. Normally, you should decide how long you want File records to be maintained in the Catalog and set the **File Retention** period to that time. Then you can either wait and see how big your Catalog gets or make a calculation assuming approximately 154 bytes for each File saved and knowing the number of Files that are saved during each backup and the number of Clients you backup.

For example, suppose you do a backup of two systems, each with 100,000 files. Suppose further that you do a Full backup weekly and an Incremental every day, and that the Incremental backup typically saves 4,000 files. The size of your database after a month can roughly be calculated as:

```
   Size = 154 * No. Systems * (100,000 * 4 + 10,000 * 26)
```

where we have assumed 4 weeks in a month and 26 incremental backups per month. This would give the following:

```
   Size = 154 * 2 * (100,000 * 4 + 10,000 * 26)
or
   Size = 308 * (400,000 + 260,000)
or
   Size = 203,280,000 bytes
```

So for the above two systems, we should expect to have a database size of approximately 200 Megabytes. Of course, this will vary according to how many files are actually backed up.

Below are some statistics for a MySQL database containing Job records for five Clients beginning September 2001 through May 2002 (8.5 months) and File records for the last 80 days. (Older File records have been pruned). For these systems, only the user files and system files that change are backed up. The core part of the system is assumed to be easily reloaded from the RedHat rpms.

In the list below, the files (corresponding to Bacula Tables) with the extension .MYD contain the data records whereas files with the extension .MYI contain indexes.

You will note that the File records (containing the file attributes) make up the large bulk of the number of records as well as the space used (459 Mega Bytes including the indexes). As a consequence, the most important Retention period will be the **File Retention** period. A quick calculation shows that for each File that is saved, the database grows by approximately 150 bytes.

```
     Size in
      Bytes   Records    File
============  =========  ===========
        168          5  Client.MYD
      3,072             Client.MYI
344,394,684  3,080,191  File.MYD
115,280,896             File.MYI
  2,590,316    106,902  Filename.MYD
  3,026,944             Filename.MYI
        184          4  FileSet.MYD
      2,048             FileSet.MYI
     49,062      1,326  JobMedia.MYD
     30,720             JobMedia.MYI
    141,752      1,378  Job.MYD
     13,312             Job.MYI
      1,004         11  Media.MYD
      3,072             Media.MYI
  1,299,512     22,233  Path.MYD
    581,632             Path.MYI
         36          1  Pool.MYD
      3,072             Pool.MYI
          5          1  Version.MYD
      1,024             Version.MYI
```

This database has a total size of approximately 450 Megabytes.

If we were using SQLite, the determination of the total database size would be much easier since it is a single file, but we would have less insight to the size of the individual tables as we have in this case.


Disaster Recovery Using Bacula

Index

Automatic Volume Recycling


Catalog Maintenance

Index

Using AutoChangers

Automatic Volume Recycling

Normally, **Bacula** will write on a volume, and once the tape is written, it may append to the volume, but it will never overwrite the data thus destroying it. When we speak of **recycling** volumes, we mean that **Bacula** can write over the previous contents of a volume. Thus all previous data will be lost.

If you are like me, you may not want **Bacula** to automatically recycle (reuse) tapes. This requires a large number of tapes, and at some point, I will manually recycle them. However, many people prefer to have a Pool of tapes that are used for daily backups and recycled once a week, another Pool of tapes that are used for Full backups once a week and recycled monthly, and finally a Pool of tapes that are used once a month and recycled after a year or two. With a scheme like this, your pool of tapes remains constant.

By properly defining your Volume Pools with appropriate Retention periods, **Bacula** can manage the recycling (such as defined above) automatically.

Automatic recycling of Volumes is controlled by three records in the **Pool** resource definition in the Director's configuration file. These three records are:

- AutoPrune = yes
- VolumeRetention = <time>
- Recycle = yes

Automatic recycling of Volumes is performed by **Bacula** only when it wants a new Volume and no appendable Volumes are available in the Pool. It will then search the Pool for any Volumes with the **Recycle** flag set and whose Volume Status is **Full**. At that point, the recycling occurs in two steps. The first is that a Volume must be purged of all Jobs and Files, and the second step is the actual recycling of the Volume. The Volume will be purged if the VolumeRetention period has expired. If no volumes can be recycled for any of the reasons stated above, **Bacula** will request operator intervention (i.e. it will ask you to label a new volume).

Automatic Pruning

By setting **AutoPrune** to **yes** you will permit **Bacula** to automatically prune all Volumes in the Pool when a Job needs another Volume. When a Job requests another volume and there are no Volumes with Volume Status **Append** available, **Bacula** will begin volume pruning. This means that all Jobs that are older than the **VolumeRetention** period will be pruned from every Volume that has Volume Status **Full** and has Recycle set to **yes**. When all files are pruned from a Volume, it will be marked as **Purged** implying that no Jobs remain on the volume. The records that control the pruning are described below.

AutoPrune = <yes/no>
: If AutoPrune is set to **yes** (default), Bacula (version 1.20 or greater) will automatically apply the Volume retention period when it needs a new Volume and no appendable volumes are available. The default is **yes**.

Volume Retention = <time−period−specification>
: The Volume Retention record defines the length of time that **Bacula** will guarantee that the Volume is not reused counting from the time the last job stored on the Volume terminated.

When this time period expires, and if **AutoPrune** is set to **yes**, and a new Volume is needed, but no appendable Volume is available, **Bacula** will prune (remove) Job records that are older than the specified Volume Retention period.

The Volume Retention period takes precedence over any Job Retention period you have specified in the Client resource. It should also be noted, that the Volume Retention period is obtained by reading the Catalog Database Media record rather than the Pool resource record. This means that if you change the VolumeRetention in the Pool resource record, you must ensure that the corresponding change is made in the catalog.

Retention periods are specified in seconds, but as a convenience, there are a number of modifiers that permit easy specification in terms of minutes, hours, days, weeks, months, quarters, or years on the record. See the Configuration chapter of this manual for additional details of modifier specification.

The default is 1 year.

Recycle = <yes/no>

If Recycle is set to **no** (the default), then even if **Bacula** prunes all the Jobs on the volume, it will not consider the tape for recycling. If Recycle is set to **yes** and all Jobs have been pruned, the volume status will be set to **purged** and the volume may then be reused when another volume is needed. If the volume is reused, it is relabeled with the same Volume Name and all previous data will be lost.

Recycling

After all Volumes of a Pool have been pruned (as mentioned above, this happens when a Job needs a new Volume and no appendable Volumes are available), **Bacula** will look for the oldest Volume that is Purged (all Jobs and Files expired), and if the **Recycle** flag is on (Recycle=yes) for that Volume, **Bacula** will relabel it and write new data on it.

Recycle Status

Each Volume inherits the Recycle status (yes or no) from the Pool resource record when the Media record is created (normally when the Volume is labeled). This Recycle status is stored in the Media record of the Catalog. Using the the Console program, you may subsequently change the Recycle status for each Volume. For example in the following output from **list volumes**:

```
+------------+-----------+-----------+----------+-----------------+----------+-------+
| VolumeName | MediaType | VolStatus | VolBytes | LastWritten     | VolReten | Recyc |
+------------+-----------+-----------+----------+-----------------+----------+-------+
| File0001   | File      | Full      | 4190055  | 2002-05-25 18:42 | 14400    | 1     |
| File0002   | File      | Full      | 1896460  | 2002-05-26 18:05 | 14400    | 1     |
| File0003   | File      | Full      | 1896460  | 2002-05-26 20:05 | 14400    | 1     |
| File0004   | File      | Full      | 1896460  | 2002-05-26 21:35 | 14400    | 1     |
| File0005   | File      | Full      | 1896460  | 2002-05-26 22:05 | 14400    | 1     |
| File0006   | File      | Full      | 1896460  | 2002-05-26 19:35 | 14400    | 1     |
| File0007   | File      | Purged    | 1896466  | 2002-05-26 18:05 | 14400    | 1     |
+------------+-----------+-----------+----------+-----------------+----------+-------+
```

all the volumes are marked as recyclable, and the last Volume, **File0007** has been purged, so it may be immediately recycled. The other volumes are all marked recyclable and when their

Volume Retention period (14400 seconds or 4 hours) expires, they will be eligible for pruning, and possible recycling. Even though Volume **File0007** has been purged, all the data on the Volume is still recoverable. A purged Volume simply means that there are no entries in the Catalog. Even if the Volume Status is changed to **Recycle**, the data on the Volume will be recoverable. The data is lost only when the Volume is re−labeled and re−written.

To modify Volume **File0001** so that it cannot be recycled, you use the **update volume pool=File** command in the console program, or simply **update** and **Bacula** will prompt you for the information.

```
+------------+-----------+-----------+----------+-----------------+----------+-------+
| VolumeName | MediaType | VolStatus | VolBytes | LastWritten     | VolReten | Recyc |
+------------+-----------+-----------+----------+-----------------+----------+-------+
| File0001   | File      | Full      | 4190055  | 2002−05−25 18:42 | 14400    | 0     |
| File0002   | File      | Full      | 1897236  | 2002−05−26 23:35 | 14400    | 1     |
| File0003   | File      | Full      | 1896460  | 2002−05−26 20:05 | 14400    | 1     |
| File0004   | File      | Full      | 1896460  | 2002−05−26 21:35 | 14400    | 1     |
| File0005   | File      | Full      | 1896460  | 2002−05−26 22:05 | 14400    | 1     |
| File0006   | File      | Full      | 1896460  | 2002−05−26 19:35 | 14400    | 1     |
| File0007   | File      | Purged    | 1896466  | 2002−05−26 18:05 | 14400    | 1     |
+------------+-----------+-----------+----------+-----------------+----------+-------+
```

In this case, **File0001** will never be automatically recycled. The same effect can be achieved by setting the Volume Status to Read−Only.

Making Bacula Use a Single Tape

Most people will want **Bacula** to fill a tape and when it is full, a new tape will be mounted, and so on. However, as an extreme example, it is possible for **Bacula** to write on a single tape, and every night to rewrite it. To get this to work, you must do two things: first, set the VolumeRetention to less than your save period (one day), and the second item is to make **Bacula** mark the tape as full after using it once. This is done using **UseVolumeOnce = yes**. If this latter record is not used and the tape is not full after the first time it is written, **Bacula** will simply append to the tape and eventually request another volume. Using the tape only once, forces the tape to be marked **Full** after each use, and the next time **Bacula** runs, it will recycle the tape.

An example Pool resource that does this is:

```
Pool {
  Name = DDS−4
  Use Volume Once = yes
  Pool Type = Backup
  AutoPrune = yes
  VolumeRetention = 12h # expire after 12 hours
  Recycle = yes
}
```

A Daily, Weekly, Monthly Tape Usage Example

This example is meant to show you how one could define a fixed set of volumes that **Bacula** will rotate through on a regular schedule. There are an infinite number of such schemes, all of which have various advantages and disadvantages.

We start with the following assumptions:

- A single tape has more than enough capacity to do a full save.
- There are 10 tapes that are used on a daily basis for incremental backups. They are prelabled Daily1 ... Daily10.
- There are 4 tapes that are used on a weekly basis for full backups. They are labeled Week1 ... Week4.
- There are 12 tapes that are used on a monthly basis for full backups. They are numbered Month1 ... Month12
- A full backup is done every Saturday evening (tape inserted Friday evening before leaving work).
- No backups are done over the weekend (this is easy to change).
- The first Friday of each month, a Monthly tape is used for the Full backup.
- Incremental backups are done Monday − Friday (actually Tue−Fri mornings).

We start the system by doing a Full save to one of the weekly volumes or one of the monthly volumes. The next morning, we remove the tape and insert a Daily tape. Friday evening, we remove the Daily tape and insert the next tape in the Weekly series. Monday, we remove the Weekly tape and re−insert the Daily tape. On the first Friday of the next month, we insert the next Monthly tape in the series rather than a Weekly tape, then continue. When a Daily tape finally fills up, **Bacula** will request the next one in the series, and the next day when you notice the email message, you will mount it and **Bacula** will finish the unfinished incremental backup.

What does this give? Well, at any point, you will have a the last complete Full save plus several Incremental saves. For any given file your want to recover (or your whole system), you will have a copy of that file every day for at least the last 14 days. For older versions, you will have at least 3 and probably 4 Friday full saves of that file, and going back further, you will have a copy of that file made on the beginning of the month for at least a year.

So you have copies of any file (or your whole system) for at least a year, but as you go back in time, the time between copies increases from daily to weekly to monthly.

What would the **Bacula** configuration look like to implement such a scheme?

```
Schedule {
  Name = "NightlySave"
  Run = Level=Full Pool=FullSaves sat at 03:05
  Run = Level=Incremental Pool=Daily tue-fri at 03:05
}

Job {
  Name = "NightlySave"
  Type = Backup
  Level = Full
  Client=XXXXXXXXXX
  FileSet="File Set"
  Messages = Standard
  Storage = DDS-4
  Pool = FullSaves
  Schedule = "NightlySave"
}

Storage {
  Name = DDS-4
```

```
  Address = XXXXXXXXXX
  SDPort = 9103
  Password = XXXXXXXXXXXX
  Device = FileStorage
  Media Type = 8mm
}

FileSet {
  Name = "File Set"
  Include = signature=MD5 {
    ffffffffffffffff
  }
  Exclude = { *.o }
}

Pool {
  Name = Daily
  Pool Type = Backup
  AutoPrune = yes
  VolumeRetention = 10d   # recycle in 10 days
  Maximum Volumes = 10
  Recycle = yes
}

Pool {
  Name = FullSaves
  Use Volume Once = yes
  Pool Type = Backup
  AutoPrune = yes
  VolumeRetention = 30d  # recycle in 30 days (default)
  Recycle = yes
}
```

The FullSaves Pool will contain both your weekly and your monthly tapes. However, after labeling your Monthly tapes, you will need to manually set the VolumeRetention period for each tape to 1 year.

In a future version of **Bacula**, the Schedule resource will be enhanced to have 1sat, 2sat, ... to represent the first, second, etc saturday of the month. When that is available, you could separate the Weekly and the Monthly tapes into two different Pools.

Automatic Pruning and Recycling Example

Perhaps the best way to understand the various resource records that come into play during automatic pruning and recycling is to run a Job that goes through the whole cycle. If you add the following resources to your Director's configuration file:

```
Schedule {
  Name = "30 minute cycle"
  Run = Level=Full Pool=File Messages=Standard Storage=File hourly at 0:05
  Run = Level=Full Pool=File Messages=Standard Storage=File hourly at 0:35
}

Job {
  Name = "Filetest"
  Type = Backup
  Level = Full
```

```
  Client=XXXXXXXXXX
  FileSet="Test Files"
  Messages = Standard
  Storage = File
  Pool = File
  Schedule = "30 minute cycle"
}

Storage {
  Name = File
  Address = XXXXXXXXXX
  SDPort = 9103
  Password = XXXXXXXXXXXX
  Device = FileStorage
  Media Type = File
}

FileSet {
  Name = "Test Files"
  Include = signature=MD5 {
    fffffffffffffff
  }
  Exclude = { *.o }
}

Pool {
  Name = File
  Use Volume Once = yes
  Pool Type = Backup
  LabelFormat = File
  AutoPrune = yes
  VolumeRetention = 4h
  Maximum Volumes = 12
  Recycle = yes
}
```

Where you will need to replace the **fffffffff**'s by the appropriate files to be saved for your configuration. For the FileSet Include, choose a directory that has one or two megabytes maximum since there will probably be approximately 8 copies of the directory that **Bacula** will cycle through.

In addition, you will need to add the following to your Storage daemon's configuration file:

```
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;
}
```

With the above resources, **Bacula** will start a Job every half hour that saves a copy of the directory you chose to /tmp/File0001 ... /tmp/File0012. After 4 hours, Bacula will start recycling the backup Volumes (/tmp/File0001 ...). You should see this happening in the output produced. Bacula will automatically create the Volumes (Files) the first time it uses them.

To turn it off, either delete all the resources you've added, or simply comment out the **Schedule** record in the **Job** resource.

  
Catalog Maintenance Index Using AutoChangers

  
The Bacula Console Restore Command Index Catalog Maintenance

Disaster Recovery Using Bacula

General

When disaster strikes, you must have a plan, and you must have prepared in advance otherwise the work of recovering your system and your files will be considerably greater. For example, if you have not previously saved the partitioning information for your hard disk, how can you properly rebuild it if the disk must be replaced?

Unfortunately, many of the steps one must take before and immediately after a disaster are very operating system dependent. As a consequence, this chapter will discuss in detail disaster recovery (also called Bare Metal Recovery) for **Linux** and **Solaris**. For Solaris, the procedures are still quite manual. For FreeBSD the same procedures may be used but they are not yet developed. For Win32, no luck. Apparently an "emergency boot" disk allowing access to the full system API without interference does not exist.

Important Considerations

Here are a few important considerations concerning disaster recovery that you should take into account before a disaster strikes.

- If the building which houses your computers burns down or is otherwise destroyed, do you have off−site backup data?
- Disaster recovery is much easier if you have several machines. If you have a single machine, how will you handle unforeseen events if your only machine is down?
- Do you want to protect your whole system and use Bacula to recover everything? or do you want to try to restore your system from the original installation disks and apply any other updates and only restore user files?

Bare Metal Recovery on Linux

The remainder of this section concerns recovering a **Linux** computer, and parts of it relate to the Red Hat version of Linux. The **Solaris** procedures can be found below.

A so called "Bare Metal" recovery is one where you start with an empty hard disk and you restore your machine. There are also cases where you may lose a file or a directory and want it restored. Please see the previous chapter for more details for those cases.

Bare Metal Recovery assumes that you have the following four items for your system:

- An emergency boot disk allowing you to boot without a hard disk
- A Bacula Rescue Disk containing your disk information and a number of helpful scripts (described below)
- A full Bacula backup of your system possibly including Incremental or Differential backups since the last Full backup
- A second system running the Bacula Director, the Catalog, and the Storage daemon. (this is not an absolute requirement, but how to get around it is not yet documented here)

Restrictions

In addition, to the above assumptions, the following conditions or restrictions apply:

- Linux only −− tested only on Red Hat, but should work on other Linuxes
- The scripts handle only SCSI and IDE disks
- All partitions will be recreated, but only **ext2**, **ext3**, and **swap** partitions will be reformatted. Any other partitions such as Windows FAT partitions will not be formatted by the scripts, but you can do it by hand
- You are using either **lilo** or **grub** as a boot loader, and you know which one (not automatically detected)

Scripts

The scripts discussed below can be found in the **rescue/linux** subdirectory of the Bacula source code.

Preparation for a Bare Metal Recovery

There are two things you should do immediately on all (Linux) systems for which you wish to do a bare metal recovery:

1. Create a system emergency boot disk or alternatively a system installation boot floppy. This step can be skipped if you have an Installation CDROM and your machine will boot from CDROM (most modern computers will).
2. Create a Bacula Rescue disk, which captures the current working state of your computer and creates scripts to restore it.

Creating an Emergency Boot Disk Here you have several choices:

- Create a tomsrtbt disk (any Linux)
- Create an emergency boot disk (any Linux I think)
- Create a Red Hat Installation disk (Red Hat specific)
- Others

tomsrtbt

My preference is to create and use a **tomsrtbt** emergency boot disk because it gives you a very clean Linux environment (with a 2.2 kernel) and the most utilities. See http://www.toms.net/rb/ for more details on this. It is very easy to do and well worth the effort. However, I recommend that you create both especially if you have non−standard hardware. You may find that **tomsrtbt** will not work with your network driver (he surely has one, but you must explicitly put it on the disk), whereas the Linux rescue is more likely to work.

Emergency Boot Disk

To create a standard Linux emergency boot disk you must first know the name of the kernel, which you can find with:

```
ls −l /boot
```

and looking on the **vmlinux−...** line or alternative do an

```
uname −a
```

then become root and with a blank floppy in the drive, enter the following command:

```
mkbootdisk −−device /dev/fd0 2.4.18−18
```

where you replace "2.4.18−18" by your system name.

This disk can then be booted and you will be in an environment with a number of important tools available. Some disadvantages of this environment as opposed to **tomsrtbt** are that you must enter **linux rescue** at the boot prompt or the boot will fail without a hard disk; it requires a disk boot image or a CDROM to be mounted, if the CDROM is released, you will loose a large number of the tools.

Red Hat Installation Disk

Specific to Red Hat Linux, is to create an Installation floppy, which can also be used as an emergency boot disk. The advantage of this method is that it works in conjunction with the installation CDROM and hence during the first part of restoring the system, you have a much larger number of tools available (on the CDROM). This can be extremely useful if you are not sure what really happened and you need to examine your system in detail.

To make a Red Hat Linux installation disk, do the following:

```
mount the Installation CDROM (/mnt/cdrom)
cd /mnt/cdrom/images
dd if=boot.img of=/dev/fd0 bs=1440k
```

Now that you have either an emergency boot disk or an installation floppy, you will be able to reboot your system in the absence of your hard disk or with a damaged hard disk. This method has the same disadvantages compared to **tomsrtbt** disk as mentioned above for the Emergency Boot Disk.

Creating a Bacula Rescue Disk

Simply having a boot disk is not sufficient to re−create things as they were. To solve this problem, we will create a Bacula Rescue disk. Everything that will be written to this disk will first be placed into the **<bacula−src>/rescue/linux** directory.

The first step is while your system is up and running normally, you use a **Bacula** script called **getdiskinfo** to capture certain important information about your hard disk configuration (partitioning, formatting, mount points, ...). **getdiskinfo** will also create a number of scripts using the information found that can be used in an emergency to repartition your disks, reformat them, and restore a statically linked version of the **Bacula** file daemon so that your disk can be restored from within a minimal boot environment.

The first step is to run **getdiskinfo** as follows:

```
su
cd <bacula-src>/rescue/linux
./getdiskinfo
```

getdiskinfo works for either IDE or SCSI drives and recognizes both ext2 and ext3 file systems. If you wish to restore other file systems, you will need to modify the code. This script can be run multiple times, but really only needs to be run once unless you change your hard disk configuration.

Assuming you have a single hard disk on device /dev/hda, **getdiskinfo** will create the following files:

partition.hda
: This file contains the shell commands to repartition your hard disk drive /dev/hda to the current state. If you have additional drives (e.g. /dev/hdc), you will find one of these files for each drive. DO NOT EXECUTE THIS SCRIPT UNLESS YOU WANT YOUR HARD DISK REPARTITIONED

format.hda
: This file contains the shell commands that will format each of the partitions on your hard drive. It knows about ext2, ext3, and swap partitions. All other partitions, you must manually format. It is recommended that any Microsoft partitions be partitioned with Microsoft's **format** command rather than using Unix tools. DO NOT EXECUTE THIS SCRIPT UNLESS YOU WANT YOUR HARD DISK REFORMATTED

mount_drives
: This script will mount all ext2 and ext3 drives that were previously mounted. They will be mounted on
/mnt/drive/. This is used just before running the statically linked Bacula so that it can access your drives for the restore.

restore_bacula
: This script will restore the File daemon from the Bacula Rescue disk. Building the Bacula Rescue disk will be described later. This will provide your emergency boot environment with a Bacula file daemon.

start_network
: This script will start your network using the simplest possible commands. You will need to verify that the IP address used in this script is correct. In addition, if you have several ethernet cards, you may need to make other modifications to this script.

sfdisk
: This is the program that will repartition your hard disk, and it is normally found in **/sbin/sfdisk**. It is placed in this directory so that it will be included on the rescue disk as it is not normally available with all emergency boot environments.

sfdisk.gz
: This is the version of sfdisk that works with **tomsrtbt**. The standard sfdisk described above will not run under tomsrtbt.

The **getdiskinfo** program (actually a shell script) will also create a subdirectory named **diskinfo**, which contains the following files:

```
df.bsi
disks.bsi
fstab.bsi
ifconfig.bsi
```

```
mount.bsi
mount.ext2.bsi
mount.ext3.bsi
mtab.bsi
route.bsi
sfdisk.disks.bsi
sfdisk.hda.bsi
sfdisk.make.hda.bsi
```

Each of these files contains some important piece of information (sometimes redundant) about your hard disk setup or your network. Normally, you will not need this information, but it will be written to the Bacula Rescue disk just in case. Since it is normally not used, we will leave it to you to examine those files at your leisure.

Building a Static File Daemon

The second of the three steps in creating your Bacula Rescue disk is to build a static version of the File daemon. Do so by either configuring Bacula as follows or by allowing the **make_rescue_disk** script described below make it for you:

```
cd <bacula-src>
./configure <normal-options> --enable-static-fd
make
cd src/filed
strip bacula-fd
cp bacula-fd ../../rescue/linux
cp bacula-fd.conf ../../rescue/linux
```

Finally, in <bacula−src>/rescue/linux, ensure that the WorkingDirectory, PIDDirectory, and SubSysDirectory all point to reasonable locations on a stripped down system. If you are using **tomsrtbt** you will also want to replace machine names with IP addresses since there is no resolver running. With the Linux Rescue disk, network address mapping seems to work. Don't forget that at the time this version of the Bacula File daemon runs, your file system will not be restored. In my bacula−fd.conf, I use **/var/working**.

Writing the Bacula Rescue Floppy

When you have everything you need (output of getdiskinfo, Bacula File daemon, ...), you create your rescue floppy by putting a blank tape into your floppy disk drive and entering:

```
su
./make_rescue_disk
```

This script will reformat the floppy and write everything in the current directory and all files in the **diskinfo** directory to the floppy. If you supply the appropriate command line options, it will also build a static version of the Bacula file daemon and copy it along with the configuration file to the disk. Also using a command line option, you can make it write a compressed tar file containing all the files whose names are in **backup.etc.list** to the floppy. The list as provided contains names of files in **/etc** that you might need in a disaster situation. It is not needed, but in some cases such as a complex network setup, you may find it useful.

Options for make_rescue_disk

The following command line options are available for the make_rescue_disk script:

```
Usage: make_rescue_disk
  -h, --help             print this message
  --make-static-bacula   make static File daemon and add to diskette
  --copy-static-bacula   copy static File daemon to diskette
  --copy-etc-files       copy files in etc list to diskette
```

Briefly the options are:

--make-static-bacula
: If this option is specified, the script will assume that you have already configured and built Bacula. It will then proceed to build a statically linked version and copy it along with bacula-fd.conf to the current directory, then write it to the rescue disk.

--copy-static-bacula,/dt>
: If this option is given, the script will assume that you already have a copy of the statically linked Bacula in the current directory named **bacula-fd** as well as the configuration script. They will then be written to the rescue disk.

--copy-etc-files
: If this option is specified, the script will tar the files in **backup.etc.list** and write them to the rescue disk.

Please examine the contents of the rescue floppy to ensure that it has everything you want and need. If not modify the scripts as necessary and re-run it until it is correct.

Now that you have both a system boot floppy and a Bacula Rescue floppy, assuming you have a full backup of your system made by Bacula, you are ready to handle nearly any kind of emergency restoration situation.

Restoring Your System

Now, let's assume that your hard disk has just died and that you have replaced it with an new identical drive. In addition, we assume that you have:

1. A recent Bacula backup (Full plus Incrementals)
2. An emergency boot floppy (preferably **tomsrtbt**)
3. A Bacula Rescue Disk
4. Your Bacula Director, Catalog, and Storage daemon running on another machine on your local network.

This is a relatively simple case, and later in this chapter, as time permits, we will discuss how you might recover from a situation where the machine that crashes is your main Bacula server (i.e. has the Director, the Catalog, and the Storage daemon).

You will take the following steps to get your system back up and running:

1. Boot with your Emergency Floppy
2. Mount your Bacula Rescue floppy
3. Start the Network (local network)

4. Re−partition your hard disk(s) as it was before
5. Re−format your partitions
6. Restore the Bacula File daemon (static version)
7. Perform a Bacula restore of all your files
8. Re−install your boot loader
9. Reboot

Now for the details ...

Boot with your Emergency Floppy

First you will boot with your emergency floppy. If you use the Installation floppy described above, when you get to the boot prompt:

```
boot:
```

you enter **linux rescue**.

If you are booting from **tomsrtbt** simply enter the default responses.

When your machine finishes booting, you should be at the command prompt possibly with your hard disk mounted on **/mount/sysimage** (Linux emergency only). To see what is actually mounted, use:

```
df
```

Mount your Bacula Rescue Floppy

Make sure that the mount point **/mnt/floppy** exists. If not, enter:

```
mkdir −p /mnt/floppy
```

the mount your **Bacula Rescue** disk and cd to it with:

```
mount /dev/fd0 /mnt/floppy
cd /mnt/floppy
```

To simplify running the scripts make sure the current directory is on your path by:

```
PATH=$PATH:.
```

Start the Network

At this point, you should bring up your network. Normally, this is quite simple and requires just a few commands. To simplify your task, we have created a script that should work in most cases by typing:

```
./start_network
```

You can test it by pinging another machine, or pinging your broken machine machine from another machine. Do not proceed until your network is up.

Unmount Your Hard Disk (if mounted)

When you are sure you want to repartition your disk, normally, if your disk was damaged or if you are using **tomsrtbt** your hard disk will not be mounted. However, if it is you must first unmount it so that it is not in use. Do so by entering **df** and then enter the correct commands to unmount the disks. For example:

```
umount /mnt/sysimage/boot
umount /mnt/sysimage/usr
umount /mnt/sysimage/proc
umount /mnt/sysimage/
```

where you explicitly unmount (**umount**) each sysimage partition and finally, the last one being the root. Do another **df** command to be sure you successfully unmount all the sysimage partitions.

This is necessary because **sfdisk** will refuse to partition a disk that is currently mounted. As mentioned, this should never be necessary with **tomsrtbt**.

Partition Your Hard Disk(s)

If you are using **tomsrtbt**, you will need to do the following steps to get the correct sfdisk:

```
rm −f sfdisk
bzip2 −d sfdisk.bz2
```

Do not do the above steps if you are using a standard Linux boot disk.

Then proceed with partitioning your hard disk by:

```
./partition.hda
```

If you have multiple disks, do the same for each of them. For SCSI disks, the repartition script will be named: **partition.sda**. If the script complains about the disk being in use, simply go back and redo the **df** command and **umount** commands until you no longer have your hard disk mounted. Note, in many cases, if your hard disk was seriously damaged or a new one installed, it will not automatically be mounted. If it is mounted, it is because the emergency kernel found one or more possibly valid partitions.

If for some reason this proceedure does not work, you can use the information in **partition.hda** to re−partition your disks by hand using **fdisk**.

Format Your Hard Disk(s)

After partitioning your disk, you must format it appropriately. The formatting script will put back swap partitions, normal Unix partitions (ext2) and journaled partitions (ext3). Do so by entering for each disk:

```
./format.hda
```

The format script will ask you if you want a block check done. We recommend to answer yes, but realize that for very large disks this can take hours.

Mount the Newly Formatted Disks

Once the disks are partitioned and formatted, you can remount them with the **mount_drives** script. All your drives must be mounted for Bacula to be able to access them. Run the script as follows:

```
./mount_drives
df
```

The **df** will tell you if the drives are mounted. If not, re−run the script again. It isn't always easy to figure out and create the mount points and the mounts in the proper order, so repeating the **./mount_drives** command will not cause any harm and will most likely work the second time. If not, correct it by hand before continuing.

Unmount the CDROM

Next, if you are using the Red Hat installation disk, unmount the CDROM drive by doing:

```
umount /mnt/cdrom
```

This is not necessary if you are running **tomsrtbt**. In doing this, I find it is always busy, and I haven't figured out how to unmount it (Linux boot only).

Restore and Start the File Daemon

Now, change (cd) to some directory where you want to put the image of the Bacula File daemon. I use the root directory my hard disk (mounted as **/mnt/disk**) because it is easy. Then install into the current directory Bacula by running the **restore_bacula** script from the floppy drive. For example:

```
cd /mnt/disk
mkdir −p /mnt/disk/
mkdir −p /mnt/disk/working
/mnt/floppy/restore_bacula
ls −l
```

Make sure **bacula−fd** and **bacula−fd.conf** are both there.

Edit the Bacula configuration file, create the working/pid/subsys directory if you haven't already done so above, and start Bacula by entering:

```
chroot /mnt/disk /bacula−fd
```

The above command starts the Bacula File daemon with your the proper root disk location (i.e. **/mnt/disk**. If Bacula does not start correct the problem and start it. You can check if it is running by entering:

```
ps fax
```

You can kill Bacula by entering:

```
kill −TERM <pid>
```

where **pid** is the first number printed in front of the first occurrence of **bacula−fd** in the **ps fax** command.

Now, you should be able to use another computer with Bacula installed to check the status by entering:

```
status client=xxxx
```

into the Console program, where xxxx is the name of the client you are restoring.

One common problem is that your **bacula−fd.conf** may contain machine addresses that are not properly resolved on this stripped down system because it is not running DNS. In that case, be prepared to edit **bacula−fd.conf** to replace the name of the Director's machine with its IP address. Or better yet, do this before building the Bacula rescue disk.

Restore Your Files

On the computer that is running the Director, you now run a **restore** command and select the files to be restored (normally everything), but before starting the restore, there is one final change you must make using the **mod** option. You must change the **Where** directory to be the root by using the **mod** option just before running the job and selecting **Where**. Set it to:

```
/
```

then run the restore.

You might be tempted to avoid using **chroot** and running Bacula directly and then using a **Where** to specify a destination of **/mnt/disk**. This is possible, however, the current version of Bacula always restores files to the new location, and thus any soft links that have been specified with absolute paths will end up with **/mnt/disk** prefixed to them. In general this is not fatal to getting your system running, but be aware that you will have to fix these links if you do not use **chroot**.

Final Step

At this point, the restore should have finished with no errors, and all your files will be restored. One last task remains and that is to write a new boot sector so that your machine will boot. For **lilo**, you enter the following command:

```
run_lilo
```

If you are using grub instead of lilo, you must enter the following:

```
run_grub
```

Note, I've had quite a number of problems with **grub** because it is rather complicated and not designed to install easily under a simplified system. So, if you experience errors or end up unexpectedly in a **chroot** shell, simply exit back to the normal shell and type in the appropriate commands from the **run_grub** script by hand until you get it to install.

Reboot

Reboot your machine by simply entering **exit** until you get to the main prompt then enter **ctl−d**.

If everything went well, you should now be back up and running. If not, re−insert the emergency boot floppy, boot, and figure out what is wrong.

At this point, you will probably want to remove the temporary copy of Bacula that you installed. Do so with:

```
rm −f /bacula−fd /bacula−fd.conf
rm −rf /working
```

Problems or Bugs

Since every flavor and every release of Linux is different, there are likely to be some small difficulties with the scripts, so please be prepared to edit them in a minimal environment. A rudimentary knowledge of **vi** is very useful. Also, these scripts do not do everything. You will need to reformat Windows partitions by hand, for example.

Getting the boot loader back can be a problem if you are using **grub** because it is so complicated. If all else fails, reboot your system from your floppy but using the restored disk image, then proceed to a reinstallation of grub (looking at the run−grub script can help). By contrast, lilo is a piece of cake.

Bugs

When performing the bare metal recovery using the Red Hat emergency boot disk (actually the installation boot disk), I was never able to release the cdrom, and when the system came up **/mnt/cdrom** was soft linked to **/mnt/disk/dev/hdd**, which is not correct. I fixed this in each case by deleting and simply remaking it with **mkdir −p /mnt/cdrom**.

tomsrtbt

This is a single floppy (1.722Meg) that really has A LOT of software. For example, by default (version 2.0.103) you get:

AHA152X AHA1542 AIC7XXX BUSLOGIC DAC960 DEC_ELCP(TULIP) EATA EEXPRESS/PRO/PRO100 EL2 EL3 EXT2 EXT3 FAT FD IDE−CD/DISK/TAPE IMM INITRD ISO9660 JOLIET LOOP MATH_EMULATION MINIX MSDOS NCR53C8XX NE2000 NFS NTFS PARPORT PCINE2K PCNET32 PLIP PPA RTL8139 SD SERIAL/_CONSOLE SLIP SMC_ULTRA SR ST VFAT VID_SELECT VORTEX WD80x3 .exrc 3c589_cs agetty ash badblocks basename boot.b buildit.s busybox bz2bzImage bzip2 cardmgr cardmgr.pid cat chain.b chattr chgrp chmod chown chroot clear clone.s cmp common config cp cpio cs cut date dd dd−lfs debugfs ddate df dhcpcd−− dirname dmesg domainname ds du dumpe2fs e2fsck echo egrep elvis ex false fdflush fdformat fdisk filesize find findsuper fmt fstab grep group gunzip gzip halt head hexdump hexedit host.conf hostname hosts httpd i82365 ifconfig ile init inittab insmod install.s issue kernel key.lst kill killall killall5 ld ld−linux length less libc libcom_err libe2p libext2fs libtermcap libuuid lilo lilo.conf ln loadkmap login ls lsattr lsmod lua luasocket man map md5sum miterm mkdir mkdosfs mke2fs mkfifo mkfs.minix mknod

mkswap more more.help mount mt mtab mv nc necho network networks nmclan_cs nslookup passwd pax pcmcia_core pcnet_cs pidof ping poweroff printf profile protocols ps pwd rc.0 rc.S rc.custom rc.custom.gz rc.pcmcia reboot rescuept reset resolv.conf rm rmdir rmmod route rsh rshd script sed serial serial_cs services setserial settings.s sh shared slattach sleep sln sort split stab strings swapoff swapon sync tail tar tcic tee telnet telnetd termcap test tomshexd tomsrtbt.FAQ touch traceroute true tune2fs umount undeb−− unpack.s unrpm−− update utmp vi vi.help view watch wc wget which xargs xirc2ps_cs yecho yes zcat

In addition, at Tom's Web Site, you can find a lot of additional kernel drivers and other software (such as **sdisk**, which is used by Bacula.

Building his floppy is a piece of cake. Simply download his .tar.gz file then:

```
− detar the .tar.gz archive
− become root
− cd to the tomsrtbt−<version> directory
− load a blank floppy with no bad sectors
− ./install.s
```

Solaris Bare Metal Recovery

The same basic techniques as described above apply to Solaris:

- the same restrictions as those given for Linux apply
- you will need to create a Bacula Rescue disk

However, during the recovery phase, the boot and disk preparation procedures are different:

- there is no need to create an emergency boot disk since it is an integrated part of the boot.
- you must partition and format your hard disk by hand following manual procedures as described in W. Curtis Preston's book "Unix Backup Recovery"

Once the disk is partitioned, formatted and mounted, you can continue with bringing up the network and reloading Bacula.

Preparing Solaris Before a Disaster

As mentioned above, before a disaster strikes, you should prepare the information needed in the case of problems. To do so, in the **rescue/solaris** subdirectory enter:

```
su
./getdiskinfo
./make_rescue_disk
```

The **getdiskinfo** script will, as in the case of Linux described above, create a subdirectory **diskinfo** containing the output from several system utilities. In addition, it will contain the output from the **SysAudit** program as described in Curtis Preston's book. This file **diskinfo/sysaudit.bsi** will contain the disk partitioning information that will allow you to manually follow the procedures in the "Unix Backup Recovery" book to repartition and format your hard disk. In addition, the **getdiskinfo** script will create a **start_network** script.

Once you have your your disks repartitioned and formatted, do the following:

- Start Your Network with the **start_network** script
- Restore the Bacula File daemon as documented above
- Perform a Bacula restore of all your files using the same commands as described above for Linux
- Re−install your boot loader using the instructions outlined in the "Unix Backup Recovery" book using installboot

Recovering a Server

Above, we considered how to recover a client machine where a valid **Bacula** server was running on another machine. However, what happens if your server goes down and you no longer have a running Director, Catalog, or Storage daemon? There are several solutions:

1. Bring up static versions of your Director, Catalog, and Storage daemon.
2. Move your server to another machine.

The first option, is very difficult because it requires you to have created a static version of the Director and the Storage daemon as well as the Catalog. If the Catalog uses MySQL, this may or may not be possible. In addition, to loading all these programs on a bare system (quite possible), you will need to make sure you have a valid driver for your tape drive.

The second suggestion is probably a much simpler solution, and one I have done myself. To do so, you might want to consider the following steps:

- If you are using MySQL, configure, build and install MySQL from source (or user rpms) on your new system.
- Load the **Bacula** source code onto your new system, configure, install it, and create the Bacula database.
- If you have a valid saved Bootstrap file as created for your damaged machine with WriteBootstrap, use it to restore the files to the damaged machine, where you have loaded a Bacula File daemon using the Bacula Rescue disk).
- If you have the Bootstrap file, you should now be back up and running, if you do not have a Bootstrap file, continue with the suggestions below.
- Using **bscan** scan the last set of backup tapes into your MySQL or SQLite database.
- Start Bacula, and using the Console **restore** command, restore the last valid copy of the Bacula database and the the Bacula configuration files.
- Move the database to the correct location.
- Start and restart Bacula, and with the full database using the Console **restore** command, restore all the files on the damaged machine, where you have loaded a Bacula File daemon using the Bacula Rescue disk.

Bugs and Other Considerations

Directory Modification and Access Times are Modified

When **Bacula** restores a directory, it first must create the directory, then it populates the directory with its files and subdirectories. The act of creating the files and subdirectories updates both the modification and access times associated with the directory itself. As a consequence, all

modification and access times of all directories will be updated to the time of the restore. This could be "corrected" by saving a list of all directories created during the restore, then when all files are restored, visit each of those directories and reset their modification and access times. This could possibly fail due to an out of memory condition −− don't forget that during a bare metal recovery, there is generally no swap file active.

I'm not too worried about this, and will probably provide restoration of exact directory modification times in a future release. If anyone feels this is more important that I do, please let me know.

Strange Bootstrap Files

If any of you look closely at the bootstrap file that is produced and used for the restore (I sure do), you will probably notice that the FileIndex item does not include all the files saved to the tape. This is because in some instances there are duplicates (especially in the case of an Incremental save), and in such circumstances, **Bacula** restores only the last of multiple copies of a file or directory.

Additional Resources

Many thanks to Charles Curley who wrote Linux Complete Backup and Recovery HOWTO for the The Linux Documentation Project. This is an excellent document on how to do Bare Metal Recovery on Linux systems, and it was this document that made me realize that Bacula could do the same thing.

You can find quite a few additional resources, both commercial and free at Storage Mountain, formerly known as Backup Central.

And finally, the O'Reilly book, "Unix Backup Recovery" by W. Curtis Preston covers virtually every backup and recovery topic including bare metal recovery for a large range of Unix systems.

The Bacula Console Restore Command | Index | Catalog Maintenance

Bacula 1.29 User's Guide
The Network Backup Solution

General

Beginning with version 1.23, **Bacula** provides autochanger support for writing tapes. In order to work with an autochanger, **Bacula** requires the services of an external program that actually commands the physical device. **Bacula** uses its own **mtx−changer** script to interface with a standalone program that actually does the tape changing. In principle, **mtx−changer** can be adapted to function with any autochanger program. The current version of **mtx−changer** works with the **mtx** program.

Current **Bacula** autochanger support does not include cleaning, stackers, or silos. However, under certain conditions, you may be able to make **Bacula** work with stackers (gravity feed and such).

Slots

Some autochangers have more than one read/write device (drive). The current implementation, assumes the autochanger has only one device. To properly address autochangers, **Bacula** must know which Volume is in each **slot** of the autochanger. Slots are where the changer cartridges reside when not loaded into the drive. **Bacula** numbers these slots from one to the number of cartridges contained in the autochanger.

For each Volume in your changer, you will, using the Console program, assign a slot. This information is kept in **Bacula's** media database along with the other data for the volume. If no slot is given, or the slot is set to zero, **Bacula** will not attempt to use the autochanger even if all the necessary configuration records are present.

Device Configuration Records

Configuration of autochangers within **Bacula** is done in the Device resource of the Storage daemon. Four records: **Auto Changer**, **Changer Device**, **Changer Command**, and **Maximum Changer Wait** control how **Bacula** uses the autochanger.

These four records, permitted in **Device** resources, are described in detail below:

Auto Changer = Yes|No
: The **Auto Changer** record specifies that the current device is or is not an autochanger. The default is **no**.

Changer Device = <device−name>
: In addition to the Archive Device name, you must specify a **Changer Device** name. This is because most autochangers are controlled through a different device than is used for reading and writing the cartridges. For example, on Linux, one normally uses the generic SCSI interface for controlling the autochanger, but the standard SCSI interface for reading and writing the tapes. On Linux, for the **Archive Device = /dev/nst0**, you would typically have **Changer Device = /dev/sg0**.

Changer Command = <command>
: This record is used to specify the external program to call and what arguments to pass to it. The command is assumed to be a standard program or shell script that can be executed by the operating system. This command is invoked each time that **Bacula** wishes to manipulate the autochanger. The following substitutions are made in the **command**

before it is sent to the operating system for execution:

```
%% = %
%a = archive device name
%c = changer device name
%f = Client's name
%j = Job name
%o = command  (loaded, load, or unload)
%s = Slot base 0
%S = Slot base 1
%v = Volume name
```

An actual example for using **mtx** with the **mtx−changer** script (part of the **Bacula** distribution) is:

```
Changer Command = "/usr/bin/mtx−changer %c %o %S"
```

Details of the three commands currently used by **Bacula** (loaded, load, unload) as well as the output expected by **Bacula** are give in the **Bacula Autochanger Interface** section below.

Maximum Changer Wait = <time>

This record is used to define the maximum amount of time that **Bacula** will wait for an autoloader to respond to a command (e.g. load). The default is set to 120 seconds. If you have a slow autoloader you may want to set it longer.

If the autoloader program fails to respond in this time, it will be killed and **Bacula** will request operator intervention.

An Example Configuration File

The following **Device** resource implements an autochanger:

```
Device {
  Name = "AutoChanger"
  Media Type = DDS−4
  Archive Device = /dev/nst0          # Normal archive device
  Changer Device = /dev/sg0           # Generic SCSI device name
  Changer Command = "/usr/bin/bacula/mtx−changer %c %o %S"
  AutoChanger = yes
  LabelMedia = no;                    # lets Bacula label unlabeled media
  AutomaticMount = yes;               # when device opened, read it
  AlwaysOpen = yes;
  Mount Anonymous Volumes = no;       # Require Volumes in Catalog order
}
```

where you will adapt the **Archive Device**, the **Changer Device**, and the path to the **Changer Command** to correspond to the values used on your system.

The above **Device** resource will work equally well for any standard tape drive (with device name **/dev/nst0**) since the extra autochanger commands will not be used unless a **slot** has been specified in the catalog record for the Volume to be used. See below for more details on the **slot**.

Specifying Slots When Labeling

If you add an **AutoChanger = yes** record to the Pool resource in your Director's configuration file, the **Bacula** Console will automatically prompt you for the slot number when you **add** or **label** tapes. Thus all stages of dealing with tapes can be totally automated.

Even though all the above configuration statements are specified and correct, **Bacula** will attempt to access the autochanger only if a **slot** is specified in the catalog Volume record (with the Volume name).

Testing the AutoChanger

Before attempting to use the autochanger with **Bacula**, it is preferable to "hand−test" that the changer works. To do so, we suggest you do the following commands (assuming that the **mtx−changer** script is installed in **/usr/bin/bacula/mtx−changer**:

Make sure Bacula is not running.

/usr/bin/bacula/mtx−changer /dev/sg0 list
: This command should print "1 2 3 4 5 6" or one number for each slot that is occupied in your changer. If an error message is printed, you must resolve the problem (e.g. try a different device name if **/dev/sg0** is incorrect. The more sophisticated autochangers will use /dev/sg1 to control /dev/nst0).

/usr/bin/bacula/mtx−changer /dev/sg0 unload
: If a tape is loaded, this should cause it to be unloaded.

/usr/bin/bacula/mtx−changer /dev/sg0 load 3
: Assuming you have a tape in slot 3, it will be loaded into the read slot (0).

/usr/bin/bacula/mtx−changer /dev/sg0 loaded
: It should print "3"

/usr/bin/bacula/mtx−changer /dev/sg0 unload

Once all the above commands work correctly, assuming that you have the right **Changer Command** in your configuration, **Bacula** should be able to operate the changer.

Using the Autochanger

Lets assume that you have properly defined the necessary Storage daemon Device records, and you have added the **Autochanger = yes** record to the Pool resource in your Director's configuration file.

Now you fill your autochanger with say six blank tapes.

What do you do to make **Bacula** access those tapes?

One strategy is to prelabel each of the tapes. Do so by starting **Bacula**, then with the Console program, enter the **label** command:

```
./console
Connecting to Director rufus:8101
1000 OK: rufus−dir Version: 1.26 (4 October 2002)
*label
```

it will then prints something like:

```
Using default Catalog name=BackupDB DB=bacula
The defined Storage resources are:
     1: AutoChanger
     2: File
Select Storage resource (1-2): 1
```

I select the autochanger (1), and it prints:

```
Enter new Volume name: TestVolume1
Enter slot (0 for none): 1
```

where I entered **TestVolume1** for the tape name, and slot **1** for the slot. It then asks:

```
Defined Pools:
     1: Default
     2: File
Select the Pool (1-2): 1
```

I select the Default pool. This will be automatically done if you only have a single pool, then **Bacula** will proceed to unload any loaded volume, load the volume in slot 1 and label it. In this example, nothing was in the drive, so it printed:

```
Connecting to Storage daemon AutoChanger at localhost:9103 ...
Sending label command ...
3903 Issuing autochanger "load slot 1" command.
3000 OK label. Volume=TestVolume1 Device=/dev/nst0
Media record for Volume=TestVolume1 successfully created.
Requesting mount AutoChanger ...
3001 Device /dev/nst0 is mounted with Volume TestVolume1
You have messages.
*
```

You may then proceed to label the other volumes. The messages will change slightly because **Bacula** will unload the volume (just labeled TestVolume1) before loading the next volume to be labeled.

Once all your Volumes are labeled, **Bacula** will automatically load them as they are needed.

To "see" how you have labeled your Volumes, simply enter the **list volumes** command from the Console program, which should print something like the following:

```
*list volumes
Using default Catalog name=BackupDB DB=bacula
Defined Pools:
     1: Default
     2: File
Select the Pool (1-2): 1

+---------+----------+-----------+-----------+----------+------------+----------+-------+------+
| MediaId | VolName  | MediaType | VolStatus | VolBytes | LstWritten | VolReten | Recyc | Slot |
+---------+----------+-----------+-----------+----------+------------+----------+-------+------+
| 1       | TestVol1 | DDS-4     | Append    | 0        | 0          | 30672000 | 0     | 1    |
| 2       | TestVol2 | DDS-4     | Append    | 0        | 0          | 30672000 | 0     | 2    |
| 3       | TestVol3 | DDS-4     | Append    | 0        | 0          | 30672000 | 0     | 3    |
```

```
| ...                                                                                 |
+---------+----------+----------+----------+----------+-----------+----------+-------+------+
```

Bacula Autochanger Interface

```
- Currently the changer commands used are:
    loaded -- returns number of slot loaded or 0
    load   -- loads a specified slot (note, some autochangers
              require a 30 second pause after this command)
    unload -- unloads the device (returns cassette to its slot)

- Other changer commands defined but not yet used:
    list   -- returns a single line containing list of slots containing a
              cassette, each slot is separated from the others by a space.
    slots  -- returns total number of slots
```


Automatic Volume Recycling

Index

Tips and Suggestions

Tips and Suggestions

Upgrading Bacula Versions

The first thing to do before upgrading from one version to another is to ensure that don't overwrite your production (current) version of **Bacula** until you have tested that the new version works.

If you have installed **Bacula** into a single directory, this is simple: simply make a copy of your **Bacula** directory.

If you have done a more typical Unix installation where the binaries are placed in one directory and the configuration files are placed in another, then the simplest way is to configure your new **Bacula** to go into a single file.

Whatever your situation may be (one of the two just described), you should probably start with the **defaultconf** script that can be found in the **examples** subdirectory. Copy this script to the main Bacula directory, modify it as necessary (there should not need to be many modifications), configure Bacula, build it, install it, then stop your production Bacula, copy all the ***.conf** files from your production Bacula directory to the test Bacula directory, start the test version, and run a few test backups. If all seems good, then you can proceed to install the new Bacula in place of or possibly over the old Bacula.

When installing a new **Bacula** you need not worry about losing the changes you made to your configuration files as the installation process will not overwrite them.

Getting Notified of Job Completion

One of the first things you should do is to ensure that you are being properly notified of the status of each Job run by Bacula, or at a minimum of each Job that terminates with an error.

Until you are completely comfortable with **Bacula**, we recommend that you send an email to yourself for each Job that is run. This is most easily accomplished by adding an email notification address in the **Messages** resource of your Director's configuration file. An email is automatically configured in the default configuration files, but you must ensure that the default **root** address is replaced by your email address.

For examples of how I (Kern) configure my system, please take a look at the **.conf** files found in the **examples** sub−directory. We recommend the following configuration (where you change the paths and email address to correspond to your setup). Note, the **mailcommand** and **operatorcommand** should be on a single line. They were split here for presentation:

```
Messages {
  Name = Standard
  mailcommand = "/home/bacula/bin/smtp −h localhost −f \"Bacula \"
                −s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "/home/bacula/bin/smtp −h localhost −f \"Bacula \"
                    −s \"Bacula: Intervention needed for %j\" %r"
  Mail = your−email−address = all, !skipped, !terminate
  append = "/home/bacula/bin/log" = all, !skipped, !terminate
  operator = your−email−address = mount
```

```
  console = all, !skipped, !saved
}
```

You will need to ensure that the **/home/bacula/bin** path on the **mailcommand** and the **operatorcommand** lines points to your **Bacula** binary directory where the **smtp** program will be installed. You will also want to ensure that the **your−email−address** is replaced by your email address, and finally, you will also need to ensure that the **/home/bacula/bin/log** points to the file where you want to log all messages.

With the above Messages resource, you will be notified by email of every Job that ran, all the output will be appended to the **log** file you specify, all output will be directed to the console program, and all mount messages will be emailed to you. Note, some messages will be sent to multiple destinations.

The form of the mailcommand is a bit complicated, but it allows you to distinguish whether the Job terminated in error or terminated normally. Please see the Mail Command section of the Messages Resource chapter of this manual for the details of the substitution characters used above.

Once you are totally comfortable with **Bacula** as I am, or if you have a large number of nightly Jobs as I do (eight), you will probably want to change the **Mail** command to **Mail On Error** which will generate an email message only if the Job terminates in error. If the Job terminates normally, no email message will be sent, but the output will still be appended to the log file as well as sent to the Console program.

Getting Notified that Bacula is Running

If like me, you have setup **Bacula** so that email is sent only when a Job has errors, as described in the previous section of this chapter, inevitably, one day, something will go wrong and **Bacula** can stall. This could be because **Bacula** crashes, which is vary rare, or more likely the network has caused **Bacula** to **hang** for some unknown reason.

To avoid this, you can use the **RunAfterJob** command in the Job resource to schedule a Job nightly, or weekly that simply emails you a message saying that **Bacula** is still running. For example, I have setup the following Job in my Director's configuration file:

```
Schedule {
  Name = "Watchdog"
  Run = Level=Full sun−sat at 6:05
}

Job {
  Name = "Watchdog"
  Type = Admin
  Client=Watchdog
  FileSet="Verify Set"
  Messages = Standard
  Storage = DLTDrive
  Pool = Default
  Schedule = "Watchdog"
  RunAfterJob = "/home/kern/bacula/bin/watchdog"
}

Client {
```

```
  Name = Watchdog
  Address = rufus
  FDPort = 9102
  Catalog = Verify
  Password = ""
  File Retention = 1d                 # 1 days
  Job Retention = 1m                  # 1 month
  AutoPrune = yes                     # Prune expired Jobs/Files
}
```

Where I established a schedule to run the Job nightly. The Job itself is type **Admin** which means that it doesn't actually do anything, and I've defined a FileSet, Pool, Storage, and Client, all of which are not really used (and probably don't need to be specified). The key aspect of this Job is the command:

```
  RunAfterJob = "/home/kern/bacula/bin/watchdog"
```

which runs my "watchdog" script. You can put anything in the watchdog scrip. In my case, I like to monitor the size of my catalog to be sure that **Bacula** is really pruning it. The following is my watchdog script:

```
cd /home/kern/mysql/var/bacula
du . * |
/home/kern/bacula/bin/smtp -f "Bacula " -h mail.yyyy.com \
                           -s "Bacula running" xxxx@yyyy.com
```

If you just wish to send yourself a message, you can do it with:

```
cd /home/kern/mysql/var/bacula
/home/kern/bacula/bin/smtp -f "Bacula " -h mail.yyyy.com \
                           -s "Bacula running" xxxx@yyyy.com <<END-OF-DATA
Bacula is still running!!!
END-OF-DATA
```

Maintaining a Valid Bootstrap File

By using a **WriteBootstrap** record in each of your Director's Job resources, you can constantly maintain a **bootstrap** file that will enable you to recover the state of your system as of the last backup.

When a Job resource has a **WriteBootstrap** record, **Bacula** will maintain the designated file (normally on another system but mounted by NSF) with up to date information necessary to restore your system. For example, in my Director's configuration file, I have the following record:

```
 Write Bootstrap = "/mnt/deuter/files/backup/client-name.bsr"
```

where I replace **client-name** by the actual name of the client that is being backed up. Thus, **Bacula** automatically maintains one file for each of my clients. The necessary bootstrap information is appended to this file during each **Incremental** backup, and the file is totally rewritten during each **Full** backup.

If you are starting off in the middle of a cycle (i.e. with Incremental backups) rather than at the beginning (with a Full backup), the **bootstrap** file will not be immediately valid as it must

always have the information from a Full backup as the first record. If you wish to synchronize your bootstrap file immediately, you can do so by running a **restore** command for the client and selecting a full restoration, but when the restore command asks for confirmation to run the restore Job, you simply reply no, then copy the bootstrap file that was written to the location specified on the **Write Bootstrap** record. The restore bootstrap file can be found in **restore.bsr** in the working directory that you defined. In the example given below for the client **rufus**, my input is shown in bold. Note, the JobId output has been partially truncated to fit on the page here:

```
(in the Console program)
*restore
First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Cancel
Select item:  (1-6): 5
The defined Client resources are:
     1: Minimatou
     2: Rufus
     3: Timmy
Select Client (File daemon) resource (1-3): 2
The defined FileSet resources are:
     1: Kerns Files
Item 1 selected automatically.

+-------+-------+----------+-----------------+---------+---------+----------+------------+
| JobId | Level | JobFiles | StartTime       | VolName | StrtFil | VolSesId | VolSesTime |
+-------+-------+----------+-----------------+---------+---------+----------+------------+
| 2     | F     | 84       | 2002-10-26 17:14 | test1   | 0       | 1        | 1035645259 |
+-------+-------+----------+-----------------+---------+---------+----------+------------+

You have selected the following JobId: 2
Building directory tree for JobId 2 ...

The defined Storage resources are:
     1: File
Item 1 selected automatically.

You are now entering file selection mode where you add and
remove files to be restored. All files are initially added.
Enter "done" to leave this mode.

cwd is: /
$ done

84 files selected to restore.

Run Restore job
JobName:    kernsrestore
Bootstrap:  /home/kern/bacula/working/restore.bsr
Where:      /tmp/bacula-restores
FileSet:    Kerns Files
```

```
Client:     Rufus
Storage:    File
JobId:      *None*
OK to run? (yes/mod/no): no
quit

(in a shell window)
cp ../working/restore.bsr /mnt/deuter/files/backup/rufus.bsr
```

Manually Recycling Tapes

Although automatic recycling of tapes is implemented in version 1.20 and later (see the Automatic Recycling of Volumes chapter of this manual), you may want to manually force reuse (recycling) of a tape.

The simplest way to manually recycle a tape is write an end of file on the physical tape and then to remove the Volume name from the Media database. To write an end of file mark on the tape:

```
(stop Bacula or unmount the drive)
mt -f /dev/nst0 rewind
mt -f /dev/nst0 weof
```

where you replace **/dev/nst0** with the appropriate device name on your system.

To remove the Media (Volume) from the database, you do the following:

```
./console
delete media
(select the correct one)
```

With **Bacula** versions prior to 1.19, the **delete** command will only delete the selected Volume from the database and not its associated Job and File records. With **Bacula** version 1.19 and later, all associated records are deleted from the database.

Please be aware that the **delete** command can be dangerous. Once it is done, there is currently no way to recover your database records without restoring your database.

After doing the above, you can use the **Console** program to **label** the tape again.

If you prefer not to use the **delete** command, after writing an end of file mark on the tape, you can simply re−**label** the tape using a different Volume name. In this case, the old database records will remain unused (generally no great harm except that your database is not consistent with the real world).

Security Considerations

Only the File daemon needs to run with root permission (so that it can access all files). As a consequence, you may run your Director, Storage daemon, and MySQL database server as non−root processes.

You should protect the **Bacula** port addresses (normally 9101, 9102, and 9103) from outside access by a firewall or other means of protection to prevent unauthorized use of your daemons.

You should ensure that the configuration files are not world readable since they contain passwords that allow access to the daemons. Anyone who can access the Director using a console program can restore any file from a backup Volume.

  
Using AutoChangers Index Utility Programs

Bacula 1.29 User's Guide Chapter 14

  
Tips and Suggestions Index When Bacula crashes (Kaboom)

Volume Utility Tools

This document is describes the utility programs written to aid Bacula users and developers in dealing with Volumes external to Bacula.

Specifying the Configuration File

Starting with version 1.27, each of the following programs requires a valid Storage daemon configuration file (actually, the only part of the configuration file that these programs need is the **Device** resource definitions). This permits the programs to find the configuration parameters for your archive device (generally a tape drive). By default, they read **bacula−sd.conf** in the current directory, but you may specify a different configuration file using the **−c** option.

Specifying a Device Name For a Tape

Each of these programs require a **device−name** where the Volume can be found. In the case of a tape, this is the physical device name such as **/dev/nst0** or **/dev/rmt/0ubn** depending on your system. For the program to work, it must find the identical name in the Device resource of the configuration file. In the case of the tape drive you are not required to specify the volume name. You simply mount the tape and the program will read it. However, if you have multiple volumes or you want to specify the volume name (a good idea), you do so by using a **bootstrap** file.

Specifying a Device Name For a File

If you are attempting to read or write an archive file rather than a tape, the **device−name** should be the full path to the archive location including the filename. The filename (last part of the specification) will be stripped and used as the Volume name, and the path (first part before the filename) must have the same entry in the configuration file. So, the path is equivalent to the archive device name, and the filename is equivalent to the volume name.

Specifying Multiple Volumes

If you want to use a Bacula archive that spans two Volumes, and the Volumes are on tape, you will need to specify a **bootstrap** file in addition to the device name. For example, suppose you want to read tapes **tape1** and **tape2**. First construct a **bootstrap** file named say, **list.bsr** which contains:

```
Volume=test1|test2
```

where each Volume is separated by a vertical bar. Then simply use:

```
./bls −b list.bsr /dev/nst0
```

In the case of Bacula Volumes that are on files, you may simply append volumes as follows:

```
./bls /tmp/test1\|test2
```

where the backslash (\) was necessary as a shell escape to permit entering the vertical bar (|).

bls

bls can be used to do an **ls** type listing of a **Bacula** tape or file. It is called:

```
Usage: bls [-d debug_level] <device-name>
       -b <file>       specify a bootstrap file
       -c <file>       specify a configuration file
       -d <level>       specify a debug level
       -e <file>       exclude list
       -i <file>       include list
       -j              list jobs
       -k              list blocks
       -L              list tape label
    (none of above)    list saved files
       -t              use default tape device
       -v              be verbose
       -?              print this message
```

For example, to list the contents of a tape:

```
./bls /dev/nst0
```

Or to list the contents of a file:

```
./bls /tmp/xxx
```

Note that, in the case of a file, the Volume name becomes the filename, so in the above example, you will replace the **xxx** with the name of the volume (file) you wrote.

Normally if no options are specified, **bls** will produce the equivalent output to the **ls -l** command for each file on the tape. Using other options listed above, it is possible to display only the Job records, only the tape blocks, etc. For example:

```
./bls /tmp/File002
bls: butil.c:148 Using device: /tmp
drwxrwxr-x   3 kern  kern     4096 2002-10-19 21:08  /home/kern/bacula/k/src/dird/
drwxrwxr-x   2 kern  kern     4096 2002-10-10 18:59  /home/kern/bacula/k/src/dird/CVS/
-rw-rw-r--   1 kern  kern       54 2002-07-06 18:02  /home/kern/bacula/k/src/dird/CVS/Root
-rw-rw-r--   1 kern  kern       16 2002-07-06 18:02  /home/kern/bacula/k/src/dird/CVS/Repository
-rw-rw-r--   1 kern  kern     1783 2002-10-10 18:59  /home/kern/bacula/k/src/dird/CVS/Entries
-rw-rw-r--   1 kern  kern    97506 2002-10-18 21:07  /home/kern/bacula/k/src/dird/Makefile
-rw-r--r--   1 kern  kern     3513 2002-10-18 21:02  /home/kern/bacula/k/src/dird/Makefile.in
-rw-rw-r--   1 kern  kern     4669 2002-07-06 18:02  /home/kern/bacula/k/src/dird/README-config
-rw-r--r--   1 kern  kern     4391 2002-09-14 16:51  /home/kern/bacula/k/src/dird/authenticate.c
-rw-r--r--   1 kern  kern     3609 2002-07-07 16:41  /home/kern/bacula/k/src/dird/autoprune.c
-rw-rw-r--   1 kern  kern     4418 2002-10-18 21:03  /home/kern/bacula/k/src/dird/bacula-dir.conf
...
-rw-rw-r--   1 kern  kern       83 2002-08-31 19:19  /home/kern/bacula/k/src/dird/.cvsignore
bls: Got EOF on device /tmp
84 files found.
```

Listing Bacula Jobs

If you are listing a Volume to determine what you Jobs to restore, normally the **−j** option provides you with most of what you will need as long as you don't have multiple clients. For example,

```
./bls −j /tmp/test1
Volume Record: SessId=2 SessTime=1033762386 JobId=0 DataLen=144
Begin Session Record: SessId=2 SessTime=1033762386 JobId=1 Level=F Type=B
End Session Record: SessId=2 SessTime=1033762386 JobId=1 Level=F Type=B
Begin Session Record: SessId=3 SessTime=1033762386 JobId=2 Level=I Type=B
End Session Record: SessId=3 SessTime=1033762386 JobId=2 Level=I Type=B
Begin Session Record: SessId=4 SessTime=1033762386 JobId=3 Level=I Type=B
End Session Record: SessId=4 SessTime=1033762386 JobId=3 Level=I Type=B
bls: Got EOF on device /tmp
```

shows a full save followed by two incremental saves.

Adding the **−v** option will display virtually all information that is available for each record:

Listing Bacula Blocks

Normally except for debug purposes, you will not need to list Bacula blocks (the "primitive" unit of Bacula data on the Volume). However, you can do so with:

```
./bls −k /tmp/File002
bls: butil.c:148 Using device: /tmp
Block: 1 size=64512
Block: 2 size=64512
...
Block: 65 size=64512
Block: 66 size=19195
bls: Got EOF on device /tmp
End of File on device
```

By adding the **−v** option, you can get more information, which can be useful in knowing what sessions were written to the volume:

```
./bls −k −v /tmp/File002

Volume Label:
Id                : Bacula 0.9 mortal
VerNo             : 10
VolName           : File002
PrevVolName       :
VolFile           : 0
LabelType         : VOL_LABEL
LabelSize         : 147
PoolName          : Default
MediaType         : File
PoolType          : Backup
HostName          :
Date label written: 2002−10−19 at 21:16

Block: 1 blen=64512 First rec FI=VOL_LABEL SessId=1 SessTim=1035062102 Strm=0 rlen=147
Block: 2 blen=64512 First rec FI=6 SessId=1 SessTim=1035062102 Strm=DATA rlen=4087
```

```
Block: 3 blen=64512 First rec FI=12 SessId=1 SessTim=1035062102 Strm=DATA rlen=5902
Block: 4 blen=64512 First rec FI=19 SessId=1 SessTim=1035062102 Strm=DATA rlen=28382
...
Block: 65 blen=64512 First rec FI=83 SessId=1 SessTim=1035062102 Strm=DATA rlen=1873
Block: 66 blen=19195 First rec FI=83 SessId=1 SessTim=1035062102 Strm=DATA rlen=2973
bls: Got EOF on device /tmp
End of File on device
```

Armed with the SessionId and the SessionTime, you can extract just about anything.

If you want to know even more, add a second **−v** to the command line to get a dump of every record in every block.

```
./bls −k −v −v /tmp/File002
bls: block.c:79 Dump block  80f8ad0: size=64512 BlkNum=1
               Hdrcksum=b1bdfd6d cksum=b1bdfd6d
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=VOL_LABEL Strm=0 len=147 p=80f8b40
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=SOS_LABEL Strm=−7 len=122 p=80f8be7
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=1 Strm=UATTR len=86 p=80f8c75
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=2 Strm=UATTR len=90 p=80f8cdf
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=3 Strm=UATTR len=92 p=80f8d4d
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=3 Strm=DATA len=54 p=80f8dbd
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=3 Strm=MD5 len=16 p=80f8e07
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=4 Strm=UATTR len=98 p=80f8e2b
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=4 Strm=DATA len=16 p=80f8ea1
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=4 Strm=MD5 len=16 p=80f8ec5
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=5 Strm=UATTR len=96 p=80f8ee9
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=5 Strm=DATA len=1783 p=80f8f5d
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=5 Strm=MD5 len=16 p=80f9668
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=UATTR len=95 p=80f968c
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=32768 p=80f96ff
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=32768 p=8101713
bls: block.c:79 Dump block  80f8ad0: size=64512 BlkNum=2
               Hdrcksum=9acc1e7f cksum=9acc1e7f
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=contDATA len=4087 p=80f8b40
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=DATA len=31970 p=80f9b4b
bls: block.c:92    Rec: VId=1 VT=1035062102 FI=6 Strm=MD5 len=16 p=8101841
...
```

bextract

Normally, you will restore files by running a **Restore** Job from the **Console** program. However, **bextract** can be used to extract a single file or a list of files from a **Bacula** tape or file. In fact, **bextract** can be a useful tool to restore files to an empty system assuming you are able to boot, you have statically linked **bextract** and you have an appropriate **bootstrap** file.

It is called:

```
Usage: bextract [-d debug_level] <device-name> <directory-to-store-files>
       -b        specify a bootstrap file
       -dnn            set debug level to nn
       -e <file>       exclude list
       -i <file>       include list
       -?              print this message
```

where **device-name** is the Archive Device (raw device name or full filename) of the device to be read, and **directory-to-store-files** is a path prefix to prepend to all the files restored.

NOTE: On Windows systems, if you specify a prefix of say d:/tmp, any file that would have been restored to **c:/My Documents** will be restored to **d:/tmp/My Documents**. That is the original drive specification will be stripped. If no prefix is specified, the file will be restored to the original drive.

Extracting with Include or Exclude Lists

Using the **-e** option, you can specify a file containing a list of files to be excluded. Wildcards can be used in the exclusion list. This option will normally be used in conjunction with the **-i** option (see below). Both the **-e** and the **-i** options may be specified at the same time as the **-b** option. The bootstrap filters will be applied first, then the include list, then the exclude list.

Likewise, and probably more importantly, with the **-i** option, you can specify a file that contains a list (one file per line) of files and directories to include for restoration. The list must contain the full filename with the path. If you specify a path name only, it should be terminated with a slash, and all files and subdirectories of that path will be restored. If you specify a line containing only the filename (e.g. **my-file.txt**) it probably will not be extracted because you have not specified the full path.

For example, if the file **include-list** contains:

```
/home/kern/bacula
/usr/local/bin
```

N.B. Please do not include a trailing slash on directory names. This confuses **Bacula**.

Then the command:

```
./bextract -i include-list /dev/nst0 /tmp
```

will restore from the Bacula archive **/dev/nst0** all files and directories in the backup from **/home/kern/bacula** and from **/usr/local/bin**. The restored files will be placed in a file of the

original name under the directory **/tmp** (i.e. /tmp/home/kern/bacula/... and /tmp/usr/local/bin/...).

Extracting With a Bootstrap File

The **−b** option is used to specify a **bootstrap** file containing the information needed to restore precisely the files you want. Specifying a **bootstrap** file is optional but recommended because it gives you the most control over which files will be restored. For more details on the **bootstrap** file, please see Restoring Files with the Bootstrap File chapter of this document. Note, you may also use a bootstrap file produced by the **restore** command. For example:

```
./bextract −b bootstrap−file /dev/nst0 /tmp
```

The bootstrap file allows detailed specification of what files you want restored (extracted). You may specify a bootstrap file and include and/or exclude files at the same time. The bootstrap conditions will first be applied, and then each file record seen will be compared to the include and exclude lists.

Extracting From Multiple Volumes

If you wish to extract files that span several Volumes, you can specify the Volume names in the bootstrap file or you may specify the Volume names on the command line by separating them with a vertical bar. See the section above under the **bls** program entitled **Listing Multiple Volumes** for more information. The same techniques apply equally well to the **bextract** program.

bscan

The **bscan** program can be used to re−create a database (catalog) from the backup information written to one or more Volumes. With some care, it can also be used to synchronize your catalog with a Volume (not currently recommended as it is untested).

After the loss of a hard disk, if you do not have a valid **bootstrap** file for reloading your system, you may want to consider using **bscan** to re−create your database, which can then be used to **restore** your system to its previous state.

It is called:

```
Usage: bscan [options]
       −b bootstrap      specify a bootstrap file
       −dnn              set debug level to nn
       −m                update media info in database
       −n name           specify the database name (default bacula)
       −u user           specify database user name (default bacula)
       −p password       specify database password (default none)
       −r                list records
       −s                synchronize or store in database
       −v                verbose
       −w dir            specify working directory (default /tmp)
       −?                print this message
```

If you are using SQLite as your database, you must supply the **−w** option and provide the path to the "working−directory" which contains the database (normally bacula.db).

If you are using MySQL, there is no need to supply a working directory since MySQL knows where its databases are.

If you have provided security on your MySQL database, you may also need to supply either the database name (**−b** option), the user name (**−u** option), and/or the password (**−p**) options.

Using bscan to Compare a Volume to an existing Catalog

If you wish to compare the contents of a Volume to an existing catalog without changing the catalog, you can safely do so if and only if you do **not** specify either the **−m** or the **−s** options. However, at this time (Bacula version 1.26), the comparison routines are not as good or as thorough as they should be, so we don't particularly recommend this mode other than for testing.

Using bscan to Create a new Catalog from a Volume

This is the mode for which **bscan** is most useful. Normally, you should start wit a freshly created catalog that contains no data. For example, use the **./drop_bacula_tables** and **./make_bacula_tables** before running **bscan**.

PLEASE TAKE CARE NOT TO DELETE A VALID DATABASE IN USING **./drop_bacula_tables**.

Starting with a fresh database and say a single Volume named **TestVolume1**, you would create a bootstrap file (**bootstrap.bsr**) that contains:

```
Volume="TestVolume1"
```

If there are multiple volumes, add them to the first line of the bootstrap file separated by vertical bar symbols (|). Then you would run a command such as:

```
./bscan −b bootstrap.bsr −v −s −w /usr/bin/bacula/working /dev/nst0
```

In the above, the bootstrap file (**−b**) was specified to define the volume or volumes to scan, the **−v** option for verbose output (this can be omitted if desired), the **−s** option that tells **bscan** to store information in the database, the location of your working−directory (**−w** option) as you previously defined it, and finally the physical device name **/dev/nst0**.

For example, after having done a full backup of a directory, then two incrementals, I reinitialized the SQLite database as described above, and using the bootstrap.bsr file noted above, I entered the following command:

```
./bscan −b bootstrap.bsr −v −s −w /home/kern/bacula/working /dev/nst0
```

which produced the following output:

```
bscan: bscan.c:182 Using Database: bacula, User: bacula
bscan: bscan.c:673 Created Pool record for Pool: Default
bscan: bscan.c:271 Pool type "Backup" is OK.
bscan: bscan.c:632 Created Media record for Volume: TestVolume1
bscan: bscan.c:298 Media type "DDS−4" is OK.
bscan: bscan.c:307 VOL_LABEL: OK for Volume: TestVolume1
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=1 record for original JobId=2
bscan: bscan.c:717 Created FileSet record "Kerns Files"
bscan: bscan.c:819 Updated Job termination record for new JobId=1
bscan: bscan.c:905 Created JobMedia record JobId 1, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=2 record for original JobId=3
bscan: bscan.c:708 Fileset "Kerns Files" already exists.
bscan: bscan.c:819 Updated Job termination record for new JobId=2
bscan: bscan.c:905 Created JobMedia record JobId 2, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:693 Created Client record for Client: Rufus
bscan: bscan.c:769 Created new JobId=3 record for original JobId=4
bscan: bscan.c:708 Fileset "Kerns Files" already exists.
bscan: bscan.c:819 Updated Job termination record for new JobId=3
bscan: bscan.c:905 Created JobMedia record JobId 3, MediaId 1
bscan: Got EOF on device /dev/nst0
bscan: bscan.c:652 Updated Media record at end of Volume: TestVolume1
bscan: bscan.c:428 End of Volume. VolFiles=3 VolBlocks=57 VolBytes=10,027,437
```

The key points to note are that **bscan** prints a line when each major record is created. Due to the volume of output, it does not print a line for each file record unless you supply the **−v** option twice or more on the command line.

In the case of a Job record, the new JobId will not normally be the same as the original Jobid. For example, for the first JobId above, the new JobId is 1 but the original JobId is 2. This is nothing to be concerned about as it is the normal nature of databases. **bscan** will keep everything straight.

Although **bscan** claims that it created a Client record for Client: Rufus three times, it was actually only created the first time. This is normal.

You will also notice that it read an end of file after each Job (Got EOF on device ...). Finally the last line gives the total statistics for the bscan.

If you had added a second **−v** option to the command line, Bacula would have been even more verbose, dumping virtually all the details of each Job record it encountered.

Now if you start Bacula and enter a **list jobs** command to the console program, you will get:

```
+-------+----------+-----------------+------+-----+----------+----------+---------+
| JobId | Name     | StartTime       | Type | Lvl | JobFiles | JobBytes | JobStat |
+-------+----------+-----------------+------+-----+----------+----------+---------+
| 1     | kernsave | 2002-10-07 14:59 | B    | F   | 84       | 4180207  | T       |
| 2     | kernsave | 2002-10-07 15:00 | B    | I   | 15       | 2170314  | T       |
| 3     | kernsave | 2002-10-07 15:01 | B    | I   | 33       | 3662184  | T       |
+-------+----------+-----------------+------+-----+----------+----------+---------+
```

which corresponds virtually identically with what the database contained before it was re−initialized and restored with bscan. All the Jobs and Files found on the tape are restore including most of the Media record. The Volume (Media) records restored will be marked as **Full** so that they cannot be rewritten without operator intervention.

It should be noted that **bscan** cannot restore a database to the exact condition it was in previously because a lot of the less important information contained in the database is not saved to the tape. Nevertheless, the reconstruction is sufficiently complete that you can run **restore** against it and get valid results.

Using bscan to Correct the Volume File Count

If the Storage daemon crashes during a backup Job, the catalog will no be properly updated for the Volume being used at the time of the crash. This means that the Storage daemon will have written say 20 files on the tape, but the catalog record for the Volume indicates only 19 files.

Currently, Bacula simply complains about this situation but in a future version (later than 1.26), Bacula will refuse to write on a tape that contains a different number of files from what is in the catalog. To correct this situation, you may run a **bscan** with the **−m** option (but **without** the **−s** option) to update only the final Media record for the Volumes read. This feature is implemented but has not yet been tested.

bcopy

The **bcopy** program can be used to copy one **Bacula** archive file to another. For example, you may copy a tape to a file, a file to a tape, a file to a file, or a tape to a tape. For tape to tape, you will need two tape drives. (a later version is planned that will buffer it to disk).

bcopy Command Options

```
Usage: bcopy [-d debug_level]
       -b bootstrap      specify a bootstrap file
       -c          specify configuration file
       -dnn              set debug level to nn
       -v                verbose
       -w dir            specify working directory (default /tmp)
       -?                print this message
```

By using a **bootstrap** file, you can copy parts of a **Bacula** archive file to another archive.

One of the objectives of this program is to be able to recover as much data as possible from a damaged tape. However, the current version does not yet have this feature.

As this is a new program, any feedback on its use would be appreciated.

btape

This program permits a number of elementary tape operations via a tty command interface. The **test** command, described below, can be very useful for testing older tape drive compatibility problems. Aside from initial testing of tape drive compatibility with **Bacula**, **btape** will be mostly used by developers writing new tape drivers.

btape can be dangerous to use with existing **Bacula** tapes because it will relabel a tape or write on the tape if so requested regardless that the tape may contain valuable data, so please be careful and use it only on blank tapes.

To work properly, **btape** needs to read the Storage daemon's configuration file. As a default, it will look for **bacula−sd.conf** in the current directory. If your configuration file is elsewhere, please use the **−c** option to specify where.

The physical device name must be specified on the command line, and that this same device name must be present in the Storage daemon's configuration file read by **btape**

```
Usage: btape [−c config_file] [−d debug_level] [device_name]
       −c <file>   set configuration file to file
       −dnn        set debug level to nn
       −s          turn off signals
       −t          open the default tape device
       −?          print this message.
```

Using btape to Verify your Tape Drive

An important reason for this program is to ensure that a Storage daemon configuration file is defined so that **Bacula** will correctly read and write tapes.

It is highly recommended that you run the **test** command before running your first **Bacula** job to ensure that the parameters you have defined for your storage device (tape drive) will permit **Bacula** to function properly. You only need to mount a blank tape, enter the command, and the output should be reasonably self explanatory. For example:

```
(ensure that Bacula is not running)
./btape −c /usr/bin/bacula/bacula−sd.conf /dev/nst0
```

The output will be:

```
Tape block granularity is 1024 bytes.
btape: btape.c:376 Using device: /dev/nst0
*
```

Enter the test command:

```
test
```

The output produced should be something similar to the following:

```
btape: btape.c:652 Append files test.
```

```
I'm going to write one record  in file 0,
                  two records in file 1,
             and three records in file 2

btape: btape.c:410 Rewound /dev/nst0
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:440 Wrote EOF to /dev/nst0
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:440 Wrote EOF to /dev/nst0
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:440 Wrote EOF to /dev/nst0
btape: btape.c:410 Rewound /dev/nst0
btape: btape.c:668 Now moving to end of media.
btape: btape.c:530 Moved to end of media
btape: btape.c:670
We should be in file 3. I am at file 3. This is correct! <========

btape: btape.c:673
Now I am going to attempt to append to the tape.
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:440 Wrote EOF to /dev/nst0
btape: btape.c:410 Rewound /dev/nst0
1 block of 64512 bytes in file 0                        <========
End of File mark.
2 blocks of 64512 bytes in file 1                       <========
End of File mark.
3 blocks of 64512 bytes in file 2                       <========
End of File mark.
1 block of 64512 bytes in file 3                        <========
End of File mark.
End of File mark.
End of tape
Total files=4, blocks=7, bytes = 451584                 <========
btape: btape.c:679 The above scan should have four files of:
One record, two records, three records, and one record respectively.

btape: btape.c:683 Append block test.

I'm going to write a block, an EOF, rewind, go to EOM,
then backspace over the EOF and attempt to append a second block in the first file.

btape: btape.c:410 Rewound /dev/nst0
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:440 Wrote EOF to /dev/nst0
btape: btape.c:410 Rewound /dev/nst0
btape: btape.c:530 Moved to end of media
btape: btape.c:693 We should be at file 1. I am at EOM File=1.
                   This is correct! <======
btape: btape.c:695 Doing backspace file.
btape: btape.c:556 Back spaced one file.
```

```
btape: btape.c:697 Write second block, hoping to append to first file.
btape: btape.c:782 Wrote one record of 32768 bytes.
btape: btape.c:790 Wrote block to device.
btape: btape.c:440 Wrote EOF to /dev/nst0
btape: btape.c:410 Rewound /dev/nst0
btape: btape.c:701 Done writing, scanning results
2 blocks of 64512 bytes in file 0
End of File mark.
End of File mark.
End of tape
Total files=1, blocks=2, bytes = 129024                  <========
btape: btape.c:703 The above should have one file of two blocks.
```

If the output put is not essentially identical to the above, please resolve the problem(s) before attempting to use **Bacula**. I have manually indicated the most important lines with <======, but please review the output carefully.

Tips for Resolving Problems

Incorrect File Number

When **Bacula** moves to the end of the medium, it normally uses the **ioctl(MTEOM)** function. Then **Bacula** uses the **ioctl(MTIOCGET)** function to retrieve the current file position from the **mt_fileno** field. Some SCSI tape drivers will use a fast means of seeking to the end of the medium and in doing so, they will not know the current file position and hence return a **−1**. As a consequence, if you get **"This is NOT correct!"** in the positioning tests, this may be the cause. You must correct this condition in order for **Bacula** to work.

There are two possible solutions to the above problem of incorrect file number:

- Figure out how to configure your SCSI driver to keep track of the file position during the MTEOM request. This is the preferred solution.
- Modify the **Device** resource of your **bacula−sd.conf** file to include:

```
Hardware End of File = no
```

This will cause **Bacula** to use the MTFSF request to seek to the end of the medium, and **Bacula** will keep track of the file number itself.

Incorrect Number of Blocks

Bacula's preferred method of working with tape drives (sequential devices) is to run in variable block mode. All modern tape drives support this mode, but some older drives (in particular the QIC drives) as well as the ATAPI ide−scsi driver run only in fixed block mode.

Even in variable block mode, with the exception of the first record on the second or subsequent volume of a multi−volume backup, **Bacula** will write blocks of a fixed size. However, in reading a tape, **Bacula** will assume that for each read request, exactly one block from the tape will be transferred. This the most common way that tape drives work and is well supported by **Bacula**.

Drives that run in fixed block mode can cause serious problems for **Bacula** if the drive's block size does not correspond exactly to **Bacula's** block size. In fixed block size mode, drivers may

transmit a partial block or multiple blocks for a single read request. From **Bacula's** point of view, this destroys the concept of tape blocks. In order for **Bacula** to run in fixed block mode, you must include the following records in the Storage daemon's Device resource definition:

```
Minimum Block Size = nnn
Maximum Block Size = nnn
```

where **nnn** must be the same for both records and must be identical to the driver's fixed block size.

We recommend that you avoid this configuration if at all possible. In any case, as of version 1.27, it is not at all clear that Bacula's handling of these fixed block drivers really works.

Ensuring that the Tape Modes Are Properly Set

If you have a modern SCSI tape drive and you are having problems with the **test** command as noted above, it may be that some program has set one or more of the your SCSI driver's options to non−default values. For example, if your driver is set to work in SysV manner, Bacula will not work correctly because it expects BSD behavior. To reset your tape drive to the default values, you can try the following:

```
become super user
mt −f /dev/nst0 rewind
mt −f /dev/nst0 stoptions buffer−writes async−writes read−ahead
```

The above command will clear all options and then set those specified. None of the specified options are required by Bacula, but a number of other options such as SysV behavior must not be set.

Using btape to Simulate Bacula Filling a Tape

Because there are often problems with certain tape drives or systems when end of tape conditions occur, **btape** has a special command **fill** that causes it to write random data to a tape until the tape fills. It then writes at least one more **Bacula** block to a second tape. Finally, it reads back both tapes to ensure that the data has been written in a way that **Bacula** can recover it.

This can be an extremely time consuming process (here is is about 6 hours), and you must have two blank tapes available. As this command is completely new, it is not well tested, and has considerable room for improvement, especially during the error checking in the read−back phase.

To begin this test, you enter the **fill** command and follow the instructions.

btape Commands

The full list of commands are:

```
  Command    Description
  =======    ===========
  bsf        backspace file
  bsr        backspace record
  cap        list device capabilities
  clear      clear tape errors
```

```
eod         go to end of Bacula data for append
test        General test Bacula tape functions
eom         go to the physical end of medium
fill        fill tape, write onto second volume
unfill      read filled tape
fsf         forward space a file
fsr         forward space a record
help        print this command
label       write a Bacula label to the tape
load        load a tape
quit        quit btape
rd          read tape
readlabel   read and print the Bacula tape label
rectest     test record handling functions
rewind      rewind the tape
scan        read tape block by block to EOT and report
status      print tape status
weof        write an EOF on the tape
wr          write a single record of 2048 bytes
```

The most useful commands are:

- test −− test writing records and EOF marks and reading them back.
- fill −− completely fill a volume with records, then write a few records on a second volume, and finally, both volumes will be read back.
- readlabel −− read and dump the label on a Bacula tape.
- cap −− list the device capabilities as defined in the configuration file and as perceived by the Storage daemon.

The **readlabel** command can be used to display the details of a **Bacula** tape label. This can be useful if the physical tape label was lost or damaged.

In the event that you want to relabel a **Bacula**, you can simply use the **label** command which will write over any existing label. However, please note for labeling tapes, we recommend that you use the **label** command in the **Console** program since it will never overwrite a valid **Bacula** tape.

Other Programs The following programs are general utility programs and in general do not need a configuration file nor a device name.

smtp

smtp is a simple mail transport program that permits more flexibility than the standard mail programs typically found on Unix systems. It can even be used on Windows machines.

It is called:

```
Usage: smtp [-f from] [-h mailhost] [-s subject] [-c copy] [recipient ...]
       -c          set the Cc: field
       -dnn        set debug level to nn
       -f          set the From: field
       -h          use mailhost:port as the SMTP server
       -s          set the Subject: field
       -?          print this message.
```

If the **-f** option is not specified, **smtp** will use your userid. If the option is not specified **smtp** will use the value in the environment variable **SMTPSERVER** or if there is none **localhost**. By default the port 25 is used.

recipients is a space separated list of email recipients.

The body of the email message is read from standard input.

An example of the use of **smtp** would be to put the following statement in the **Messages** resource of your **bacula-dir.conf** file. Note, these commands should appear on a single line each.

```
  mailcommand = "/home/bacula/bin/smtp -h mail.domain.com -f \"Bacula <%r>\"
                -s \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "/home/bacula/bin/smtp -h mail.domain.com -f \"Bacula <%r>\"
                  -s \"Bacula: Intervention needed for %j\" %r"
```

Where you replace **/home/bacula/bin** with the path to your **Bacula** binary directory, and you replace **mail.domain.com** with the fully qualified name of your SMTP (email) server, which normally listens on port 25. For more details on the substitution characters (e.g. %r) used in the above line, please see the documentation of the MailCommand in the Messages Resource chapter of this manual.

It is highly recommended that you test one or two cases by hand to make sure that the **mailhost** that you specified is correct and that it will accept your email requests. Since **smtp** always uses a TCP connection rather than writing in the spool file, you may find that your **from** address is being rejected because it does not contain a valid domain, or because your message is caught in your spam filtering rules. Generally, you should specify a fully qualified domain name in the **from** field.

dbcheck

dbcheck is a simple program that will search for inconsistencies in your database, and optionally fix them. The **dbcheck** program can be found in the **<bacula−source>/src/tools** directory of the source distribution. Though it is built with the make process, it is not normally "installed".

It is called:

```
Usage: dbcheck [−d debug_level] <working−directory>
                  <bacula−database> <user> <password>
       −b              batch mode
       −dnn            set debug level to nn
       −f              fix inconsistencies
       −v              verbose
       −?              print this message
```

If the **−f** option is specified, **dbcheck** will repair (**fix**) the inconsistencies it finds. Otherwise, it will report only.

If the **−b** option is specified, **dbcheck** will run in batch mode, and it will proceed to examine and fix (if −f is set) all programmed inconsistency checks. If the **−b** option is not specified, **dbcheck** will enter interactive mode and prompt with the following:

```
Hello, this is the database check/correct program.
Please select the function you want to perform.

     1) Toggle modify database flag
     2) Toggle verbose flag
     3) Eliminate duplicate Filename records
     4) Eliminate duplicate Path records
     5) Eliminate orphaned Jobmedia records
     6) Eliminate orphaned File records
     7) Eliminate orphaned Path records
     8) Eliminate orphaned Filename records
     9) Eliminate orphaned FileSet records
    10) All (3−9)
    11) Quit
Select function number:
```

By entering 1 or 2, you can toggle the modify database flag (−f option) and the verbose flag (−v). It can be helpful and reassuring to turn off the modify database flag, then select one or more of the consistency checks (items 3 through 9) to see what will be done, then toggle the modify flag on and re−run the check.

The inconsistencies examined are the following:

- Duplicate filename records. This can happen if you accidentally run two copies of **Bacula** at the same time, and they are both adding filenames simultaneously. It is a rare occurrence, but will create an inconsistent database. If this is the case, you will receive error messages during Jobs warning of duplicate database records. If you are not getting these error messages, there is no reason to run this check.
- Duplicate path records. This can happen if you accidentally run two copies of **Bacula** at the same time, and they are both adding filenames simultaneously. It is a rare occurrence, but will create an inconsistent database. See the item above for why this

occurs and how you know it is happening.

- Orphaned JobMedia records. This happens when a Job record is deleted (perhaps by a user issued SQL statement), but the corresponding JobMedia record (one for each Volume used in the Job) was not deleted. Normally, this should not happen, and even if it does, these records generally do not take much space in your database. However, by running this check, you can eliminate any such orphans.
- Orphaned File records. This happens when a Job record is deleted (perhaps by a user issued SQL statement), but the corresponding File record (one for each Volume used in the Job) was not deleted. Note, searching for these records can be **very** time consuming (i.e. it may take hours) for a large database. Normally this should not happen as **Bacula** takes care to prevent it. Just the same, this check can remove any orphaned File records. It is recommended that you run this once a year since orphaned File records can take a large amount of space in your database.
- Orphaned Path records. This condition happens any time a directory is deleted from your system and all associated Job records have been purged. During standard purging (or pruning) of Job records, **Bacula** does not check for orphaned Path records. As a consequence, over a period of time, old unused Path records will tend to accumulate and use space in your database. This check will eliminate them. It is strongly recommended that you run this check at least once a year.
- Orphaned Filename records. This condition happens any time a file is deleted from your system and all associated Job records have been purged. This can happen quite frequently as there are quite a large number of files that are created and then deleted. In addition, if you do a system update or delete an entire directory, there can be a very large number of Filename records that remain in the catalog but are no longer used.

  During standard purging (or pruning) of Job records, **Bacula** does not check for orphaned Filename records. As a consequence, over a period of time, old unused Filename records will to accumulate and use space in your database. This check will eliminate them. It is strongly recommended that you run this check at least once a year, and for large database (more than 200 Megabytes), it is probably better to run this once every 6 months.

testfind

testfind permits listing of files using the same search engine that is used for the **Include** resource in Job resources. The original use was to ensure that the search engine was correct and to print some statistics on file name and path length. However, you may find it useful to see what bacula would do with a given **Include** resource. **testfind** program can be found in the **<bacula−source>/src/tools** directory of the source distribution. Though it is built with the make process, it is not normally "installed".

It is called:

```
Usage: testfind [−d debug_level] [−] [pattern1 ...]
       −a          print extended attributes (Win32 debug)
       −dnn        set debug level to nn
       −           read pattern(s) from stdin
       −?          print this message.

Patterns are file inclusion −− normally directories.
Debug level>= 1 prints each file found.
Debug level>= 10 prints path/file for catalog.
Errors always printed.
Files/paths truncated is number with len> 255.
Truncation is only in catalog.
```

Where a pattern is any filename specification that is valid within an **Include** resource definition. If none is specified, / (the root directory) is assumed. For example:

```
./testfind /bin
```

Would print the following:

```
Dir: /bin
Reg: /bin/bash
Lnk: /bin/bash2 −> bash
Lnk: /bin/sh −> bash
Reg: /bin/cpio
Reg: /bin/ed
Lnk: /bin/red −> ed
Reg: /bin/chgrp
...
Reg: /bin/ipcalc
Reg: /bin/usleep
Reg: /bin/aumix−minimal
Reg: /bin/mt
Lnka: /bin/gawk−3.1.0 −> /bin/gawk
Reg: /bin/pgawk
Total files    : 85
Max file length: 13
Max path length: 5
Files truncated: 0
Paths truncated: 0
```

Even though **testfind** uses the same search engine as **Bacula**, each directory to be listed, must be entered as a separate command line entry or entered one line at a time to standard input if the − option was specified.

Specifying a debug level of one (i.e. **−d1**) on the command line will cause **testfind** to print the raw filenames without showing the **Bacula** internal file type, or the link (if any). Debug levels of 10 or greater cause the filename and the path to be separated using the same algorithm that is used when putting filenames into the Catalog database.


Tips and Suggestions

Index

When Bacula crashes (Kaboom)

Bacula 1.29 User's Guide Chapter 15


Utility Programs

Index

Win32 Implementation

What To Do When Bacula Crashes (Kaboom)

If you are running on a Linux system, and you have a set of working configuration files, it is very unlikely that **Bacula** will crash. As with all software, however, it is inevitable that someday, it may crash, particularly if you are running on another operating system or using a new or unusual feature.

This chapter explains what you should do if one of the three **Bacula** daemons (Director, File, Storage) crashes.

Traceback

Each of the three **Bacula** daemons has a built−in exception handler which, in case of an error, will attempt to produce a traceback. If successful the traceback will be emailed to you.

For this to work, you need to ensure that a few things are setup correctly on your system:

1. You must have an installed copy of **gdb** (the GNU debugger), and it must be on **Bacula's** path.
2. The **Bacula** installed script file **btraceback** must be in the same directory as the daemon which dies, and it must be marked as executable.
3. The script file **btraceback.gdb** must have the correct path to it specified in the **btraceback** file.
4. You must have a **mail** program which is on **Bacula's** path.

If all the above conditions are met, the daemon that crashes will produce a traceback report and email it to you. If the above conditions are not true, you may be able to correct them by editing the **btraceback** file. In doing so, you can add a correct path to the **gdb** program, correct the path to the **btraceback.gdb** file, change the **mail** program or its path, or change your email address. The key line in the **btraceback** file is:

```
gdb −quiet −batch −x /home/kern/bacula/bin/btraceback.gdb \
     $1 $2 2>1| mail −s "Bacula traceback" your−address@xxx.com
```

Since each daemon has the same traceback code, a single btraceback file is sufficient if you are running more than one daemon on a machine.

Testing The Traceback

To "manually" test the traceback feature, you simply start **Bacula** then obtain the **PID** of the main daemon thread (there are multiple threads). Unfortunately, the output had to be split to fit on this page:

```
[kern@rufus kern]$ ps fax −−columns 132 | grep bacula−dir
 2103 ?        S      0:00 /home/kern/bacula/k/src/dird/bacula−dir −c
                                 /home/kern/bacula/k/src/dird/dird.conf
 2104 ?        S      0:00  \_ /home/kern/bacula/k/src/dird/bacula−dir −c
                                 /home/kern/bacula/k/src/dird/dird.conf
 2106 ?        S      0:00      \_ /home/kern/bacula/k/src/dird/bacula−dir −c
                                 /home/kern/bacula/k/src/dird/dird.conf
 2105 ?        S      0:00      \_ /home/kern/bacula/k/src/dird/bacula−dir −c
```

```
/home/kern/bacula/k/src/dird/dird.conf
```

which in this case is 2103. Then while **Bacula** is running, you call the program giving it the path to the **Bacula** executable and the **PID**. In this case, it is:

```
./btraceback /home/kern/bacula/k/src/dird 2103
```

It should produce an email showing you the current state of the daemon (in this case the Director), and then exit leaving **Bacula** running as if nothing happened. If this is not the case, you will need to correct the problem by modifying the **btraceback** script.

Typical problems might be that **gdb** is not on the default path. Fix this by specifying the full path to it in the **btraceback** file. Another common problem is that the **mail** program doesn't work or is not on the default path. On some systems, it is preferable to use **Mail** rather than **mail**.

Getting A Traceback On Other Systems

It should be possible to produce a similar traceback on systems other than Linux, either using **gdb** or some other debugger. Solaris with **gdb** loaded works quite fine. On other systems, you will need to modify the **btraceback** program to invoke the correct debugger, and possibly correct the **btraceback.gdb** script to have appropriate commands for your debugger. If anyone succeeds in making this work with another debugger, please send us a copy of what you modified.

Manually Running Bacula Under The Debugger

If for some reason you cannot get the automatic traceback, or if you want to interactively examine the variable contents after a crash, you can run **Bacula** under the debugger. Assuming you want to run the Storage daemon under the debugger, you would do the following:

1. Start the Director and the File daemon. If the Storage daemon also starts, you will need to find its PID as shown above (ps fax | grep bacula−sd) and kill it with a command like the following:

   ```
   kill −15 PID
   ```

   where you replace **PID** by the actual value.
2. At this point, the Director and the File daemon should be running but the Storage daemon should not.
3. cd to the directory containing the Storage daemon
4. Start the Storage daemon under the debugger:

   ```
   gdb ./bacula−sd
   ```

5. Run the Storage daemon:

   ```
   run −s −f −c ./bacula−sd.conf
   ```

   You may replace the **./bacula−sd.conf** with the full path to the Storage daemon's configuration file.

6. At this point, **Bacula** will be fully operational.
7. In another shell command window, start the Console program and do what is necessary to cause **Bacula** to die.
8. When **Bacula** crashes, the **gdb** shell window will become active and **gdb** will show you the error that occurred.
9. To get a general traceback of all threads, issue the following command:

```
thread apply all bt
```

After that you can issue any debugging command.

Rejected Volumes After a Crash

Bacula keeps the number of files on each Volume in its Catalog database so that before appending to a tape, it can verify that the number of files are correct, and thus prevent overwriting valid data. If the Director or the Storage daemon crashes before the job has completed, the tape will contain one more file than is noted in the Catalog, and the next time you attempt to use the same Volume, Bacula will reject it due to a mismatch between the physical tape and the catalog.

The easiest solution to this problem is to label a new tape and start fresh. If you wish to continue appending to the current tape, you can do so by using the **update** command in the console program to change the **Volume Files** entry in the catalog. A typical sequence of events would go like the following:

```
- Bacula crashes
- You restart Bacula
```

Bacula then prints:

```
17-Jan-2003 16:45 rufus-dir: Start Backup JobId 13, Job=kernsave.2003-01-17_16.45.46
17-Jan-2003 16:45 rufus-sd: Volume test01 previously written, moving to end of data.
17-Jan-2003 16:46 rufus-sd: kernsave.2003-01-17_16.45.46 Error: I canot write on this volu
The number of files mismatch! Volume=11 Catalog=10
17-Jan-2003 16:46 rufus-sd: Job kernsave.2003-01-17_16.45.46 waiting. Cannot find any appe
Please use the "label"  command to create a new Volume for:
    Storage:      SDT-10000
    Media type:   DDS-4
    Pool:         Default
```

To get out of this situation and use the same tape, you do the following:

```
Update choice:
     1: Volume parameters
     2: Pool from resource
Choose catalog item to update (1-2): 1
Defined Pools:
     1: Default
     2: File
Select the Pool (1-2):
+---------+------------+-----------+-----------+-----------+---------------------+--------
| MediaId | VolumeName | MediaType | VolStatus | VolBytes  | LastWritten         | VolRete
+---------+------------+-----------+-----------+-----------+---------------------+--------
| 1       | test01     | DDS-4     | Error     | 352427156 | 2003-01-17 16:46:19 | 3153600
```

```
+---------+-----------+----------+----------+----------+--------------------+--------
Enter MediaId or Volume name: 1
```

First, you chose to update the Volume parameters by entering a **1**. In the volume listing that follows, notice how the VolStatus is **Error**. We will correct that after changing the Volume Files. Continuing, you respond 1,

```
Updating Volume "test01"
Parameters to modify:
     1: Volume Status
     2: Volume Retention Period
     3: Volume Use Duration
     4: Maximum Volume Jobs
     5: Maximum Volume Files
     6: Maximum Volume Bytes
     7: Recycle Flag
     8: Slot
     9: Volume Files
    10: Done
Select parameter to modify (1-10): 9
Warning changing Volume Files can result
in loss of data on your Volume

Current Volume Files is: 10
Enter new number of Files for Volume: 11
New Volume Files is: 11
Updating Volume "test01"
Parameters to modify:
     1: Volume Status
     2: Volume Retention Period
     3: Volume Use Duration
     4: Maximum Volume Jobs
     5: Maximum Volume Files
     6: Maximum Volume Bytes
     7: Recycle Flag
     8: Slot
     9: Volume Files
    10: Done
Select parameter to modify (1-10): 1
```

Here, you have selected **9** in order to update the Volume Files, then you changed it from **10** to **11**, and you now answer **1** to change the Volume Status.

```
Current Volume status is: Error
Possible Values are:
     1: Append
     2: Archive
     3: Disabled
     4: Full
     5: Used
     6: Read-Only
Choose new Volume Status (1-6): 1
New Volume status is: Append
Updating Volume "test01"
Parameters to modify:
     1: Volume Status
     2: Volume Retention Period
     3: Volume Use Duration
     4: Maximum Volume Jobs
```

```
     5: Maximum Volume Files
     6: Maximum Volume Bytes
     7: Recycle Flag
     8: Slot
     9: Volume Files
    10: Done
Select parameter to modify (1-10): 10
Selection done.
```

At this point, you have changed the Volume Files from **10** to **11** to account for the last file being written but not updated in the database, and you changed the Volume Status back to **Append**.

This was a lot of words to describe something quite simple.

The **Volume Files** option exists only in version 1.29 and later, and you should be careful using it. Generally, if you set the value to that which Bacula said is on the tape, you will be OK, especially if the value is one more than what is in the catalog.

Utility Programs Index Win32 Implementation

The Windows Version of Bacula

General

At the current time only the File daemon or Client program has been tested on Windows. As a consequence, when we speak of the Windows version of Bacula below, we are referring to the File daemon only.

The Windows version of the **Bacula** File daemon has been tested on Win95, Win98, WinMe, WinNT, and Win2000 systems. This version of **Bacula** has been built to run under the CYGWIN environment, which provides many of the features of Unix on Windows systems. It also permitted a rapid port with very few source code changes, which means that the Windows version is for the most part running code that has long proved stable on Unix systems. Even though the Win32 version of **Bacula** is a port that relies on many Unix features, it is just the same a true Windows program. When running, it is perfectly integrated with Windows and displays its icon in the system icon tray, and provides a system tray menu to obtain additional information on how **Bacula** is running (status and events dialog boxes). If so desired, it can also be stopped by using the system tray menu, though this should normally never be necessary.

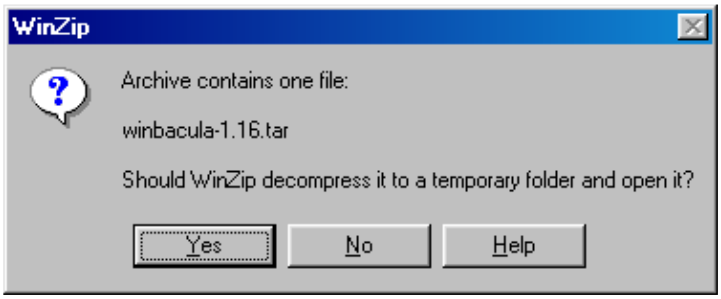
Once installed **Bacula** normally runs as a system service. This means that it is immediately started by the operating system when the system is booted, and runs in the background even if there is no user logged into the system.

Installation

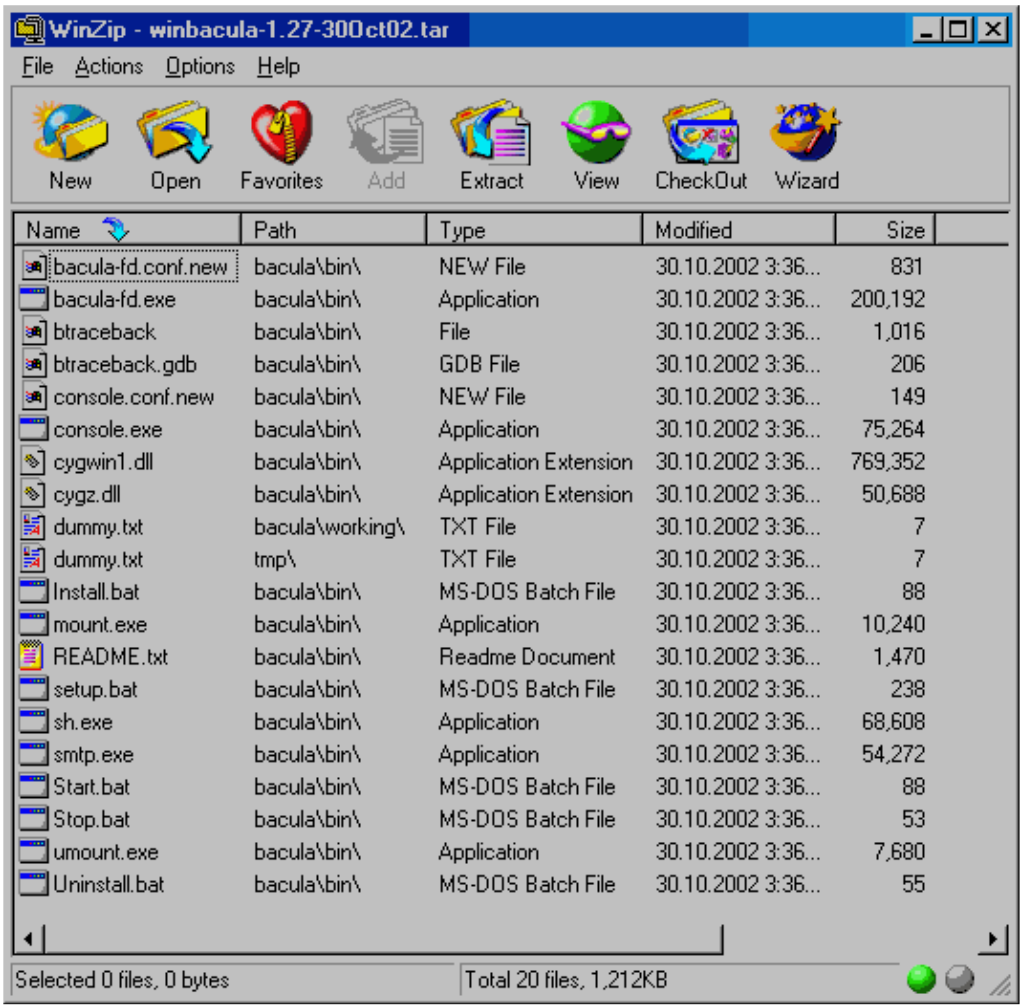
Normally, you will install the Windows version of Bacula from the binaries. This install is somewhat Unix like since you do some parts of the installation by hand. To install the binaries, you need **WinZip**.

- Simply double click on the **winbacula−1.xx.0.tar.gz** icon. The actual name of the icon will vary from one release version to another.

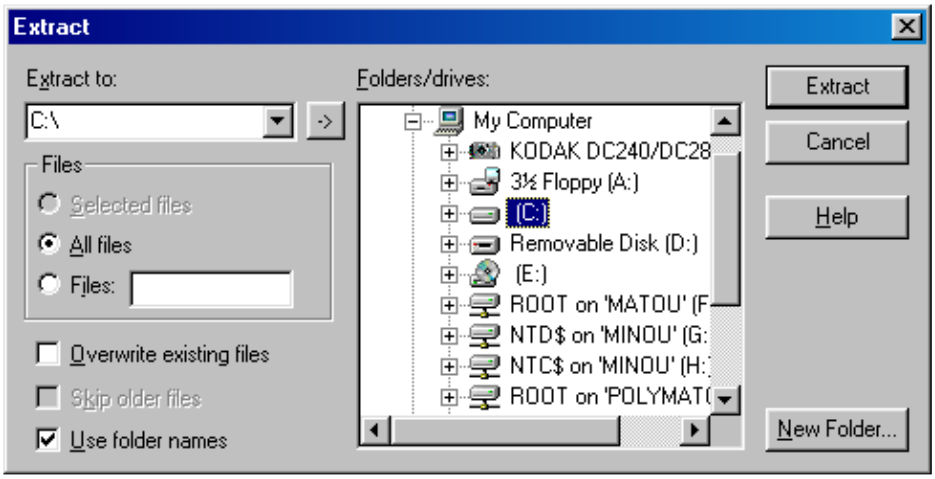
- When Zip says that it has one file and asks if it should unpack it into a temporary file, respond with **Yes**.



- You will then be presented with a WinZip dialogue that should look something like the following:



- Ensure that you extract all files and that the extraction will go into **C:**



If you wish to install the package elsewhere, please note that you will need to proceed with a manual installation, which is not particularly easy as you must rebuild the source and change the configuration file as well.

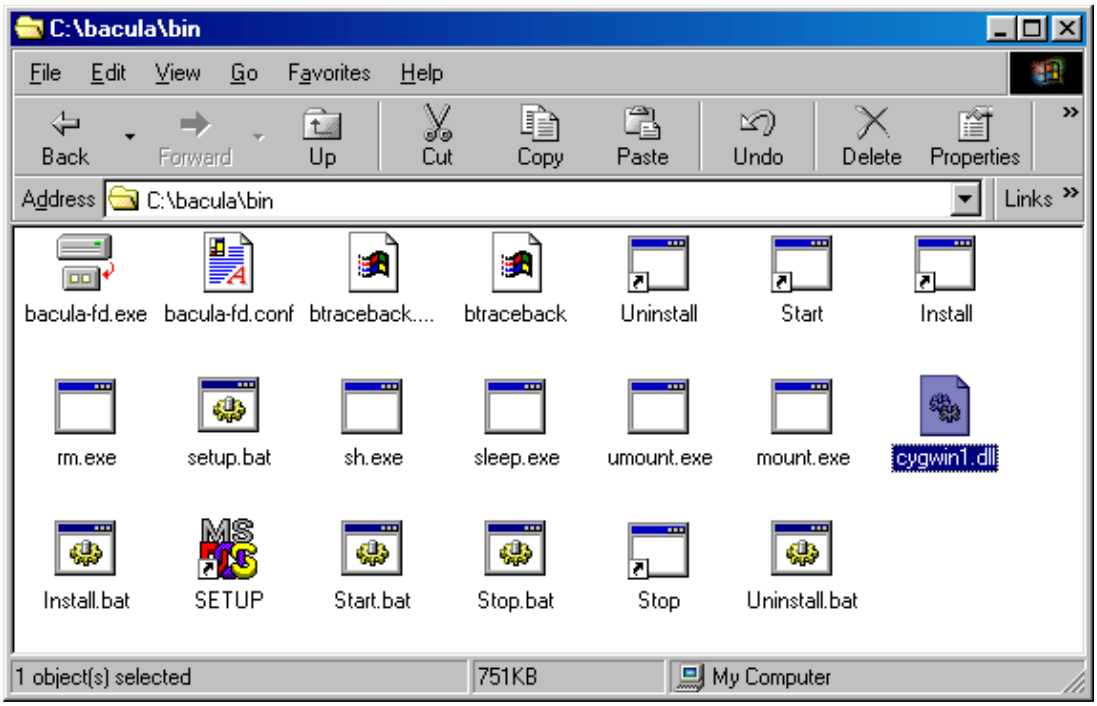
This installation assumes that you do **not** have CYGWIN installed on your computer. If you do, and you use mount points, you may need to do a special manual installation.

Once you have unzipped the binaries, open a window pointing to the binary installation folder (normally **c:\bacula**). This folder should contain additional folders such as **bin**.

Continuing the installation process:

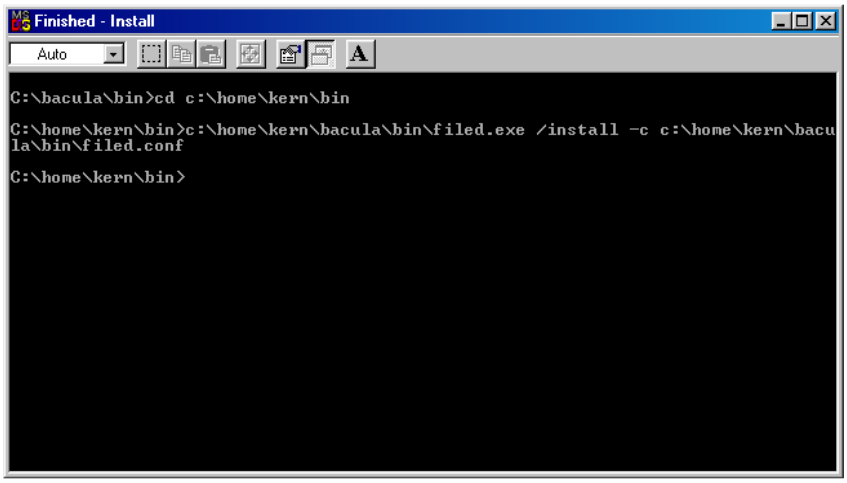
- Open the directory **c:\bacula\bin\bacula−fd.conf** in the Windows Explorer by Clicking on the **bacula** folder then on the **bin** folder. Finally double click on the file **bacula−fd.conf** and edit it to contain the values appropriate for your site. In most cases, no changes will be needed, but you probably should change the name to be something unique on your system so that you can easily distinguish messages coming from different daemons.

- To do the final step of installation, double click on the **Install.bat** program.



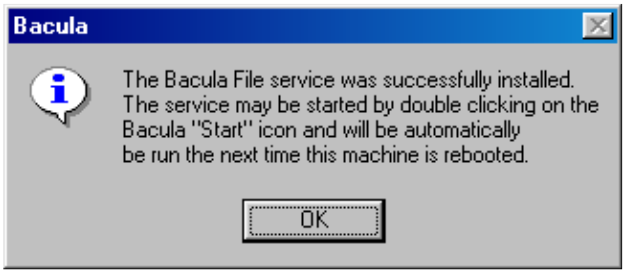
This script will setup the appropriate mount points for the directories that **Bacula** uses, it will install **Bacula** in the system registry.

If everything went well, you will get something similar to the following output in a DOS shell window:



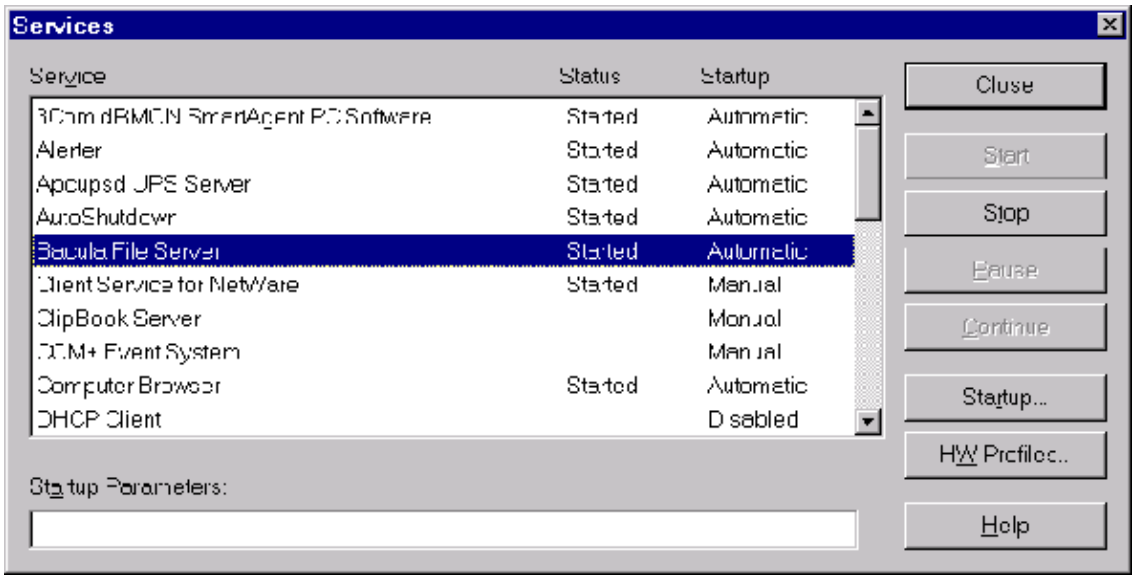
What is important to verify in the DOS window is that the root directory \ is mounted on device **c:**.

The DOS window will be followed immediately by a Windows dialog box as follows:

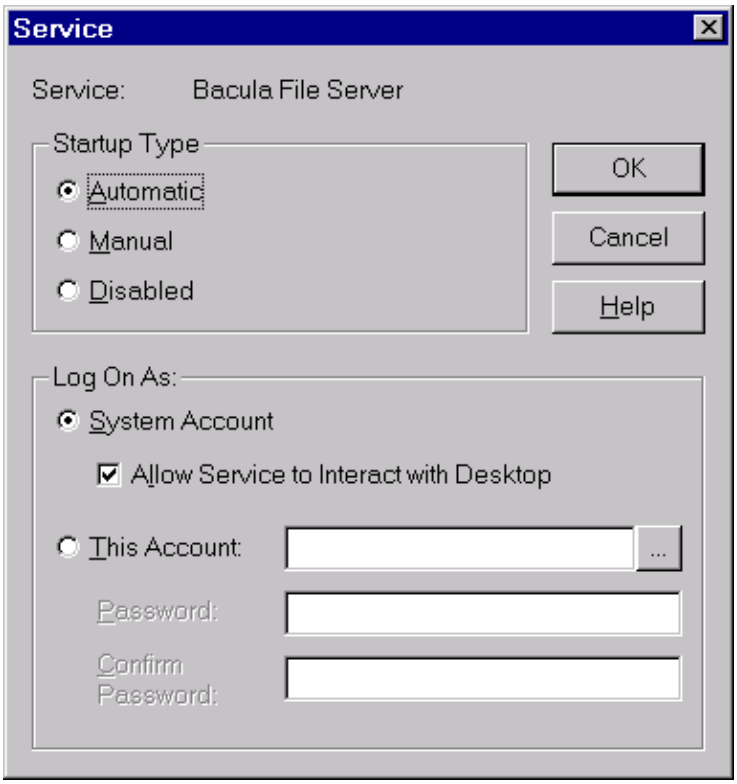


- On Windows 98, to actually start the service, either reboot the machine, which is not necessary, or double click on the **Start** icon in the **c:\bacula\bin** folder.

- On Windows NT, to start the service, either reboot the machine, which is not necessary, or go to the Control Panel, open the Services folder and start the Bacula Service by selecting the Bacula File Server:



Please ensure that the service can interact with the desktop. To do so, click on the **Sartup...** button, and you should have something similar to the following:



That should complete the installation process. When the Bacula File Server is ready to serve files, an icon representing a diskette will appear in the system tray; right click on it and a menu will appear. Select the **Events** item, and the Events dialog box should appear. There should be no error messages. By right clicking again on the system tray plug and selecting the **Status** item, you can verify that all the whether any jobs are running or not.

When the Bacula File Server begins saving files, the color of the holes in the diskette will change from white to green, and if there is an error, the holes in the diskette will change to red.

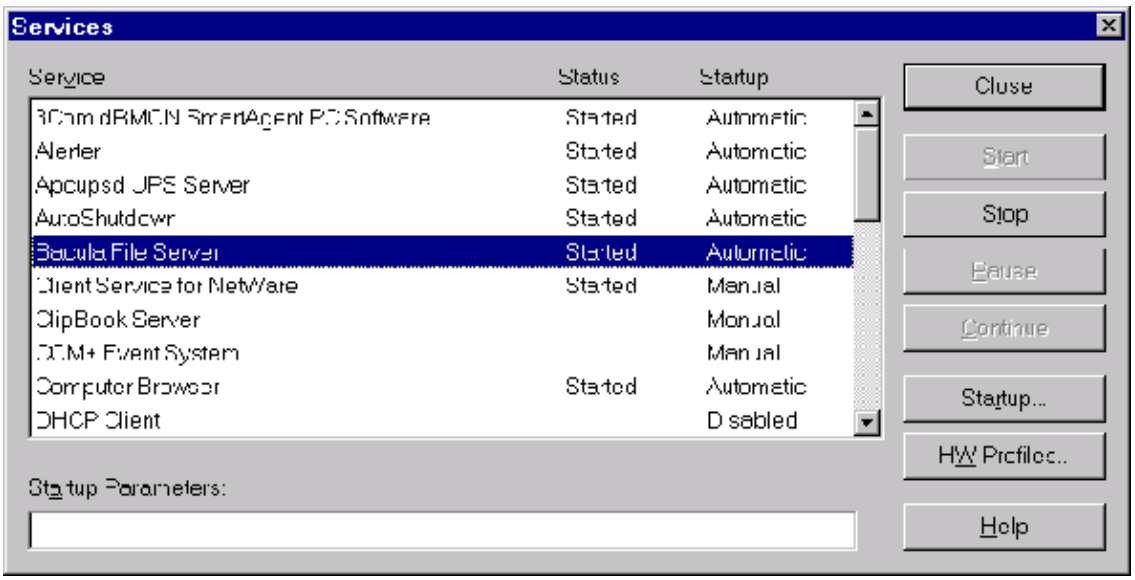
Installation Directory

The Win32 version of Bacula must reside in the **c:\Bacula** directory, and there must be a **c:\tmp** directory on your machine. The installation will do this automatically, and we recommend that you do not attempt to place **Bacula** in another directory. If you do so, you are on your own, and you will need to do a rebuild of the source.

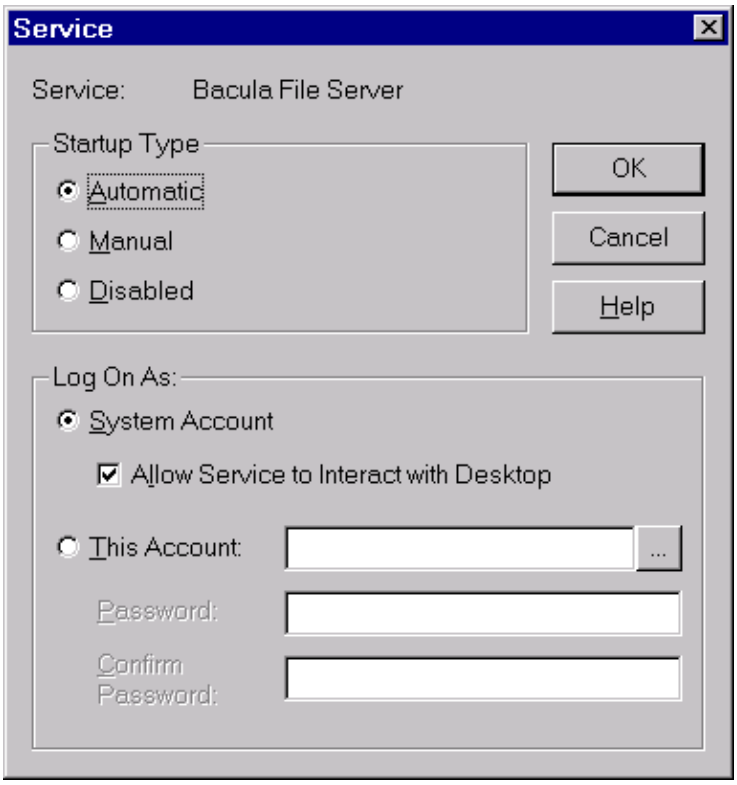
Upgrading

On Win98 and Win95 systems, to upgrade to a new release, simply stop **Bacula** by using the tray icon and selecting the **Close Bacula** menu item, or by double clicking on the **Stop** icon located in the **c:\bacula\bin** directory, then apply the upgrade and restart **Bacula**.

On WinNT systems (and Win2000 systems), you may stop **Bacula** as indicated above or alternatively you may stop **Bacula** by using the **Services** item in the **Control Panel**. Bacula bug that causes the system to prevent **bacula−fd.exe** from being overwritten even though the file is no longer being used. This is manifested by an error message when attempting load a new version and overwrite the old **bacula−fd.exe** (the extract part of WinZip as described above). Although this problem seems to be fixed, to circumvent it (if it happens to you), after shutting down the running version of **Bacula**, through the **Services** dialog in the **Control Panel**, first click on the **Stop button**:



then click on the **Startup ...** button, and in the Startup dialog select the **Disabled** button to disable **Bacula**:



After closing the dialogs, reboot the system, typical of Microsoft :−(. When the system comes back up, **Bacula** will not be automatically launched as a service, and you can install the new version. To reinstate **Bacula** as an automatic service, using the **Control Panel**: reset **Bacula** to **Automatic** startup in the Startup dialog, comment out −−− /> Then to restart **Bacula** after the new files have been loaded, go to the **Services** dialog as shown above in the installation instructions and click on **Start**. On my system, **Bacula** generally fails to start the first time after an upgrade, probably due to something remaining in the system. After receiving the error message, clicking a second time on **Start** always gets it running.

Post Installation

After installing **Bacula** and before running it, you should check the contents of **c:\bacula\bin\bacula−fd.conf** to ensure that it corresponds to your configuration.

Dealing with Problems

The most likely source of problems is authentication when the Director attempts to connect to the File daemon that you installed. This can occur if the names and the passwords defined in the File daemon's configuration file **c:\bacula\bin\bacula−fd.conf** on the Windows machine do not match with the names and the passwords in the Director's configuration file **bacula−dir.conf** located on your Unix/Linux server.

More specifically, the password found in the **Client** resource in the Director's configuration file must be the same as the password in the **Director** resource of the File daemon's configuration file. In addition, the name of the **Director** resource in the File daemon's configuration file must be the same as the name in the **Director** resource of the Director's configuration file.

It is a bit hard to explain in words, but if you understand that a Director normally has multiple Clients and a Client (or File daemon) may permit access by multiple Directors, you can see that the names and the passwords on both sides must match for proper authentication.

Running Unix like programs on Windows machines is a bit frustrating because the Windows command line shell (DOS Window) is rather primitive. As a consequence, it is not generally possible to see the debug information and certain error messages that Bacula prints. With a bit of work, however, it is possible. When everything else fails and you want to **see** what is going on, try the following:

```
Start a DOS shell Window.

cd c:\bacula\bin
bacula-fd -t >out
type out
```

The **-t** option will cause **Bacula** to read the configuration file, print any error messages and then exit. the > redirects the output to the file named **out**, which you can list with the **type** command.

If something is going wrong later, or you want to run **Bacula** with a debug option, you might try starting it as:

```
bacula-fd -b100 >out
```

In this case, **Bacula** will run until you explicitly stop it, which will give you a chance to connect to it from your Unix/Linux server.

Utility Functions

The directory **c:\Bacula\bin** contains six utility routines (actually .pif files) that you may find useful. They are:

```
Start
Stop
Install
Uninstall
```

Any of these utilities may be used on any system, with the exception of the Start utility, which cannot be used on WinNT and Win2000 systems. On those systems, the **Bacula** service must always be started through the **Services** sub-dialog of the **Control Panel**.

The **Install** and **Uninstall** utilities install and uninstall **Bacula** from the system registry only. All other pieces (files) of **Bacula** remain intact. It is not absolutely necessary for **Bacula** to be installed in the registry as it can run as a regular program. However, if it is not installed in the registry, it cannot be run as a service.

Command Line Options Specific to the Windows Version

These options are not normally seen or used by the user, and are documented here only for information purposes. At the current time, to change the default options, you must either manually run **Bacula** or you must manually edit the system registry and modify the appropriate entries.

In order to avoid option clashes between the options necessary for **Bacula** to run on Windows and the standard **Bacula** options, all Windows specific options are signaled with a forward slash character (/), while as usual, the standard **Bacula** options are signaled with a minus (−), or a minus minus (−−). All the standard **Bacula** options can be used on the Windows version. In addition, the following Windows only options are implemented:

/servicehelper
: Run the service helper application

/service
: Start **Bacula** as a service

/run
: Run the **Bacula** application

/install
: Install **Bacula** as a service in the system registry

/remove
: Uninstall **Bacula** from the system registry

/about
: Show the **Bacula** about dialogue box

/status
: Show the **Bacula** status dialogue box

/events
: Show the **Bacula** events dialogue box (not yet implemented)

/kill
: Stop any running **Bacula**

/help
: Show the **Bacula** help dialogue box

It is important to note that under normal circumstances the user should never need to use these options as they are normally handled by the system automatically once **Bacula** is installed. However, you may note these options in some of the .pif and .bat files that have been created for your use.

Building the Win32 Version from the Source

If you have the source code, follow the standard procedures for building **Bacula** on Unix in the Installation Section of this manual. Please don't forget to look at the Win32 specific instructions.


What to do when Bacula crashes (Kaboom)


Index


Using Bacula to Improve Computer Security

Using Bacula to Improve Computer Security

Since **Bacula** maintains a catalog of files, their attributes, and MD5 signatures, it can be an ideal tool for improving computer security. This is done by making a snapshot of your system files with a **Verify** Job and then checking the current state of your system against the snapshot, on a regular basis (e.g. nightly).

The first step is to set up a **Verify** Job and to run it with:

```
Level = InitCatalog
```

The **InitCatalog** level tells **Bacula** simply get the information on the specified files and to put it into the catalog. That is your database is initialized and no comparison is done. The **InitCatalog** is normally run one time manually.

Thereafter, you will run a Verify Job on a daily (or whatever) basis with:

```
Level = Catalog
```

The **Level = Catalog** level tells **Bacula** to compare the current state of the files on the Client to the last **InitCatalog** that is stored in the catalog and to report any differences. See the example below for the format of the output.

You decide what files you want to form your "snapshot" by specifying them in a **FileSet** resource, and normally, they will be system files that do not change, or that only certain features change.

Then you decide what attributes of each file you want compared by specifying comparison options on the **Include** statements that you use in the **FileSet** resource of your **Catalog** Jobs.

The Details

In the discussion that follows, we will make reference to the Verify Configuration Example that is included below in the **A Verify Configuration Example** section. You might want to look it over now to get an idea of what it does.

The main elements consist of adding a schedule, which will normally be run daily, or perhaps more often. This is provided by the **VerifyCycle** Schedule, which runs at 5:05 in the morning every day.

Then you must define a Job, much as is done below. We recommend that the Job name contain the name of your machine as well as the word **Verify** or **Check**. In our example, we named it **MatouVerify**. This will permit you to easily identify your job when running it from the Console.

You will notice that most records of the Job are quite standard, but that the **FileSet** resource contains **verify=pins5** option in addition to the standard **signature=MD5** option. If you don't want MD5 signature comparison, and we cannot imagine why not, you can drop the **signature=MD5** and none will be computed nor stored in the catalog.

The **verify=pins5** is ignored during the **InitCatalog** Job, but is used during the subsequent **Catalog** Jobs to specify what attributes of the files should be compared to those found in the catalog. **pins5** is a reasonable set to begin with, but you may want to look at the details these and other options. They can be found in the FileSet Resource section of this manual. Briefly, however, the **p** of the **pins5** tells Verify to compare the permissions bits, the **i** is to compare inodes, the **n** causes comparison of the number of links, the **s** compares the file size, and the **5** compares the MD5 checksums (this requires the **signature=MD5** option to have been set also).

You must also specify the **Client** and the **Catalog** resources for your Verify job, but you probably already have them created for your client and do not need to recreate them, they are included in the example below for completeness.

As mentioned above, you will need to have a **FileSet** resource for the Verify job, which will have the additional **verify=pins5** option. You will want to take some care in defining the list of files to be included in your **FileSet**. Basically, you will want to include all system (or other) files that should not change on your system. If you select files, such as log files or mail files, which are constantly changing, your automatic Verify job will be constantly finding differences. The objective in forming the FileSet is to choose all unchanging important system files. Then if any of those files has changed, you will be notified, and you can determine if it changed because you loaded a new package, or because someone has broken into your computer and modified your files. The example below shows a list of files that I use on my RedHat 7.3 system. Since I didn't spend a lot of time working on it, it probably is missing a few important files (if you find one, please send it to me). On the other hand, as long as I don't load any new packages, none of these files change during normal operation of the system.

Running the Verify

The first thing you will want to do is to run an **InitCatalog** level Verify Job. This will initialize the catalog to contain the file information that will later be used as a basis for comparisons with the actual file system, thus allowing you to detect any changes (and possible intrusions into your system).

The easiest way to run the **InitCatalog** is manually with the console program by simply entering **run**. You will be presented with a list of Jobs that can be run, and you will choose the one that corresponds to your Verify Job, **MatouVerify** in this example.

```
The defined Job resources are:
     1: MatouVerify
     2: kernsrestore
     3: Filetest
     4: kernsave
Select Job resource (1-4): 1
```

Next, the console program will show you the basic parameters of the Job and ask you:

```
Run Verify job
JobName:  MatouVerify
FileSet:  Verify Set
Level:    Catalog
Client:   MatouVerify
Storage:  DLTDrive
OK to run? (yes/mod/no): mod
```

Here, you want to respond **mod** to modify the parameters because the Level is by default set to **Catalog** and we want to run an **InitCatalog** Job. After responding **mod**, the console will ask:

```
Parameters to modify:
     1: Job
     2: Level
     3: FileSet
     4: Client
     5: Storage
Select parameter to modify (1-5): 2
```

you should select number 2 to modify the **Level**, and it will display:

```
Levels:
     1: Initialize Catalog
     2: Verify from Catalog
     3: Verify Volume
     4: Verify Volume Data
Select level (1-4): 1
```

Choose item 1, and you will see the final display:

```
Run Verify job
JobName:  MatouVerify
FileSet:  Verify Set
Level:    Initcatalog
Client:   MatouVerify
Storage:  DLTDrive
OK to run? (yes/mod/no): yes
```

at which point you respond **yes**, and the Job will begin.

There after the Job will automatically start according to the schedule you have defined. If you wish to immediately verify it, you can simply run a Verify **Catalog** which will be the default. No differences should be found.

What To Do When Differences Are Found

If you have setup your messages correctly, you should be notified if there are any differences and exactly what they are. For example, below is the email received after doing an update of OpenSSH:

```
HeadMan: Start Verify JobId 83 Job=RufusVerify.2002-06-25.21:41:05
HeadMan: Verifying against Init JobId 70 run 2002-06-21 18:58:51
HeadMan: File: /etc/pam.d/sshd
HeadMan:       st_ino   differ. Cat: 4674b File: 46765
HeadMan: File: /etc/rc.d/init.d/sshd
HeadMan:       st_ino   differ. Cat: 56230 File: 56231
HeadMan: File: /etc/ssh/ssh_config
HeadMan:       st_ino   differ. Cat: 81317 File: 8131b
HeadMan:       st_size  differ. Cat: 1202 File: 1297
HeadMan:       MD5 differs.
HeadMan: File: /etc/ssh/sshd_config
HeadMan:       st_ino   differ. Cat: 81398 File: 81325
HeadMan:       st_size  differ. Cat: 1182 File: 1579
HeadMan:       MD5 differs.
```

```
HeadMan: File: /etc/ssh/ssh_config.rpmnew
HeadMan:       st_ino   differ. Cat: 812dd File: 812b3
HeadMan:       st_size  differ. Cat: 1167 File: 1114
HeadMan:       MD5 differs.
HeadMan: File: /etc/ssh/sshd_config.rpmnew
HeadMan:       st_ino   differ. Cat: 81397 File: 812dd
HeadMan:       st_size  differ. Cat: 2528 File: 2407
HeadMan:       MD5 differs.
HeadMan: File: /etc/ssh/moduli
HeadMan:       st_ino   differ. Cat: 812b3 File: 812ab
HeadMan: File: /usr/bin/scp
HeadMan:       st_ino   differ. Cat: 5e07e File: 5e343
HeadMan:       st_size  differ. Cat: 26728 File: 26952
HeadMan:       MD5 differs.
HeadMan: File: /usr/bin/ssh-keygen
HeadMan:       st_ino   differ. Cat: 5df1d File: 5e07e
HeadMan:       st_size  differ. Cat: 80488 File: 84648
HeadMan:       MD5 differs.
HeadMan: File: /usr/bin/sftp
HeadMan:       st_ino   differ. Cat: 5e2e8 File: 5df1d
HeadMan:       st_size  differ. Cat: 46952 File: 46984
HeadMan:       MD5 differs.
HeadMan: File: /usr/bin/slogin
HeadMan:       st_ino   differ. Cat: 5e359 File: 5e2e8
HeadMan: File: /usr/bin/ssh
HeadMan:       st_mode  differ. Cat: 89ed File: 81ed
HeadMan:       st_ino   differ. Cat: 5e35a File: 5e359
HeadMan:       st_size  differ. Cat: 219932 File: 234440
HeadMan:       MD5 differs.
HeadMan: File: /usr/bin/ssh-add
HeadMan:       st_ino   differ. Cat: 5e35b File: 5e35a
HeadMan:       st_size  differ. Cat: 76328 File: 81448
HeadMan:       MD5 differs.
HeadMan: File: /usr/bin/ssh-agent
HeadMan:       st_ino   differ. Cat: 5e35c File: 5e35b
HeadMan:       st_size  differ. Cat: 43208 File: 47368
HeadMan:       MD5 differs.
HeadMan: File: /usr/bin/ssh-keyscan
HeadMan:       st_ino   differ. Cat: 5e35d File: 5e96a
HeadMan:       st_size  differ. Cat: 139272 File: 151560
HeadMan:       MD5 differs.
HeadMan: 25-Jun-2002 21:41
JobId:                  83
Job:                    RufusVerify.2002-06-25.21:41:05
FileSet:                Verify Set
Verify Level:           Catalog
Client:                 RufusVerify
Start time:             25-Jun-2002 21:41
End time:               25-Jun-2002 21:41
Files Examined:         4,258
Termination:            Verify Differences
```

At this point, it was obvious that these files were modified during installation of the RPMs. If you want to be super safe, you should run a **Verify Level=Catalog** immediately before installing new software to verify that there are no differences, then run a **Verify Level=InitCatalog** immediately after the installation.

To keep the above email from being sent every night when the Verify Job runs, we simply re-run the Verify Job setting the level to **InitCatalog** (as we did above in the very beginning). This will

re−establish the current state of the system as your new basis for future comparisons. Take care that you don't do an **InitCatalog** after someone has placed a Trojan horse on your system!

If you have included in your **FileSet** a file that is changed by the normal operation of your system, you will get false matches, and you will need to modify the **FileSet** to exclude that file (or not to Include it), and then re−run the **InitCatalog**.

The FileSet that is show below is what I use on my RedHat 7.3 system. With a bit more thought, you can probably add quite a number of additional files that should be monitored.

A Verify Configuration Example

```
Schedule {
  Name = "VerifyCycle"
  Run = Level=Catalog sun−sat at 5:05
}

Job {
  Name = "MatouVerify"
  Type = Verify
  Level = Catalog                     # default level
  Client = MatouVerify
  FileSet = "Verify Set"
  Messages = Standard
  Storage = DLTDrive
  Pool = Default
  Schedule = "VerifyCycle"
}
FileSet {
  Name = "Verify Set"
  Include = verify=pins5 signature=MD5 {
     /boot
     /bin
     /sbin
     /usr/bin
     /lib
     /root/.ssh
     /home/kern/.ssh
     /var/named
     /etc/sysconfig
     /etc/ssh
     /etc/security
     /etc/exports
     /etc/rc.d/init.d
     /etc/sendmail.cf
     /etc/sysctl.conf
     /etc/services
     /etc/xinetd.d
     /etc/hosts.allow
     /etc/hosts.deny
     /etc/hosts
     /etc/modules.conf
     /etc/named.conf
     /etc/pam.d
     /etc/resolv.conf
  }
  Exclude = { }
}
```

```
Client {
  Name = MatouVerify
  Address = lmatou
  Catalog = Bacula
  Password = ""
  File Retention = 80d                # 80 days
  Job Retention = 1y                  # one year
  AutoPrune = yes                     # Prune expired Jobs/Files
}

Catalog {
  Name = Bacula
  dbname = verify; user = bacula; password = ""
}
```

Win32 Implementation Index Restoring Files with a Bootstrap File

The Bootstrap File

The information in this chapter is provided so that you may either create your own bootstrap files, or so that you can edit a bootstrap file produced by **Bacula**. However, normally the bootstrap file will be automatically created for you during the **restore** command in the Console program, or by using a **Write Bootstrap** record in your Backup Jobs, and thus you will never need to know the details of this file.

The **bootstrap** file contains ASCII information that permits precise specification of what files should be restored.

File Format

The general format of a **bootstrap** file is:

<keyword>= <value>

Where each **keyword** and the **value** specify which files to restore. More precisely the **keyword** and their **values** serve to limit which files will be restored and thus act as a filter. The absence of a keyword means that all records will be accepted.

Blank lines and lines beginning with a pound sign (#) in the bootstrap file are ignored.

There are keywords which permit filtering by Volume, Client, Job, FileIndex, Session Id, Session Time, ...

The more keywords that are specified, the more selective the specification of which files to restore will be. In fact, each keyword is **AND**ed with other keywords that may be present.

For example,

```
Volume = Test-001
VolSessionId = 1
VolSessionTime = 108927638
```

directs the Storage daemon (or the **bextract** program) to restore only those files on Volume Test−001 **AND** having VolumeSessionId equal to one **AND** having VolumeSession time equal to 108927638.

The full set of permitted keywords presented in the order in which they are matched against the Volume records are:

Volume
: The value field specifies what Volume the following commands apply to. Each Volume specification becomes the current Volume, to which all the following commands apply until a new current Volume (if any) is specified. If the Volume name contains spaces, it should be enclosed in quotes.

VolFile
: The value is a file number, a list of file numbers, or a range of file numbers numbers to match on the current Volume. The file number represents the physical file on the

Volume where the data is stored. For a tape volume, this record is used to position to the correct starting file, and once the tape is past the last specified file, reading will stop.

VolBlock
The value is a block number, a list of block numbers, or a range of block numbers numbers to match on the current Volume. The block number represents the physical block on the Volume where the data is stored. This record is currently not used.

VolSessionTime
The value specifies a Volume Session Time to be matched from the current volume.

VolSessionId
The value specifies a VolSessionId, a list of volume session ids, or a range of volume session ids to be matched from the current Volume. Each VolSessionId and VolSessionTime pair corresponds to a unique Job that is backed up on the Volume.

**JobId*
The value specifies a JobId, list of JobIds, or range of JobIds to be selected from the current Volume. Note, the JobId may not be unique if you have multiple Directors, or if you have reinitialized your database. The JobId filter works only if you do not run multiple simultaneous jobs.

**Job*
The value specifies a Job name or list of Job names to be matched on the current Volume. The Job corresponds to a unique VolSessionId and VolSessionTime pair. However, the Job is perhaps a bit more readable by humans. Standard regular expressions (wildcards) may be used to match Job names. The Job filter works only if you do not run multiple simultaneous jobs.

**Client*
The value specifies a Client name or list of Clients to will be matched on the current Volume. Standard regular expressions (wildcards) may be used to match Client names. The Client filter works only if you do not run multiple simultaneous jobs.

FileIndex
The value specifies a FileIndex, list of FileIndexes, or range of FileIndexes to be selected from the current Volume. Each file (data) stored on a Volume within a Session has a unique FileIndex. For each Session, the first file written is assigned FileIndex equal to one and incremented for each file backuped.
This for a given Volume, the triple VolSessionId, VolSessionTime, and FileIndex uniquely identifies a file stored on the Volume. Multiple copies of the same file may be stored on the same Volume, but for each file, the triple VolSessionId, VolSessionTime, and FileIndex will be unique. This triple is stored in the Catalog database for each file.

Slot
The value specifies the AutoChanger slot. There may be only a single **Slot** specification for each Volume.

Stream
The value specifies a Stream, a list of Streams, or a range of Streams to be selected from the current Volume. Unless you really know what you are doing (the internals of **Bacula**, you should avoid this specification.

**JobType*
Not yet implemented.

**JobLevel*
Not yet implemented.

The **Volume** record is a bit special in that it must be the first record. The other keyword records may appear in any order and any number following a Volume record.

Multiple Volume records may be specified in the same bootstrap file, but each one starts a new set of filter critera for the Volume.

In processing the bootstrap file within the current Volume, each filter specified by a keyword is **AND**ed with the next. Thus,

```
Volume = Test−01
Client = "My machine"
FileIndex = 1
```

will match records on Volume **Test−01 AND** Client records for **My machine AND** FileIndex equal to **one**.

Multiple occurrences of the same record are **OR**ed together. Thus,

```
Volume = Test−01
Client = "My machine"
Client = "Backup machine"
FileIndex = 1
```

will match records on Volume **Test−01 AND** (Client records for **My machine OR Backup machine**) **AND** FileIndex equal to **one**.

For integer values, you may supply a range or a list, and for all other values except Volumes, you may specify a list. A list is equivalent to multiple records of the same keyword. For example,

```
Volume = Test−01
Client = "My machine", "Backup machine"
FileIndex = 1−20, 35
```

will match records on Volume **Test−01 AND (**Client records for **My machine OR Backup machine) AND (**FileIndex 1 **OR** 2 **OR** 3 ... **OR** 20 **OR** 35**)**.

As previously mentioned above, there may be multiple Volume records in the same bootstrap file. Each new Volume definition begins a new set of filter conditions that apply to that Volume and will be **OR**ed with any other Volume definitions.

As an example, suppose we query for the current set of tapes to restore all files on Client **Rufus** using the **query** command in the console program:

```
Using default Catalog name=MySQL DB=bacula
*query
Available queries:
     1: List Job totals:
     2: List where a file is saved:
     3: List where the most recent copies of a file are saved:
     4: List total files/bytes by Job:
     5: List total files/bytes by Volume:
     6: List last 10 Full Backups for a Client:
     7: List Volumes used by selected JobId:
     8: List Volumes to Restore All Files:
Choose a query (1−8): 8

Enter Client Name: Rufus
+−−−−−−−+−−−−−−−−−−−−−−−−−+−−−−−−−−−−−+−−−−−−−−−−−+−−−−−−−−−−+−−−−−−−−−−−−+
```

```
| JobId | StartTime        | VolumeName | StartFile | VolSesId | VolSesTime |
+-------+------------------+------------+-----------+----------+------------+
| 154   | 2002-05-30 12:08 | test-02    | 0         | 1        | 1022753312 |
| 202   | 2002-06-15 10:16 | test-02    | 0         | 2        | 1024128917 |
| 203   | 2002-06-15 11:12 | test-02    | 3         | 1        | 1024132350 |
| 204   | 2002-06-18 08:11 | test-02    | 4         | 1        | 1024380678 |
+-------+------------------+------------+-----------+----------+------------+
```

The output shows us that there are four Jobs that must be restored. The first one is a Full backup, and the following three are all Incremental backups.

The following bootstrap file will restore those files:

```
Volume=test-02
VolSessionId=1
VolSessionTime=1022753312
Volume=test-02
VolSessionId=2
VolSessionTime=1024128917
Volume=test-02
VolSessionId=1
VolSessionTime=1024132350
Volume=test-02
VolSessionId=1
VolSessionTime=1024380678
```

As a final example, assume that the initial Full save spanned two Volumes. The output from **query** might look like:

```
+-------+------------------+------------+-----------+----------+------------+
| JobId | StartTime        | VolumeName | StartFile | VolSesId | VolSesTime |
+-------+------------------+------------+-----------+----------+------------+
| 242   | 2002-06-25 16:50 | File0003   | 0         | 1        | 1025016612 |
| 242   | 2002-06-25 16:50 | File0004   | 0         | 1        | 1025016612 |
| 243   | 2002-06-25 16:52 | File0005   | 0         | 2        | 1025016612 |
| 246   | 2002-06-25 19:19 | File0006   | 0         | 2        | 1025025494 |
+-------+------------------+------------+-----------+----------+------------+
```

and the following bootstrap file would restore those files:

```
Volume=File0003
VolSessionId=1
VolSessionTime=1025016612
Volume=File0004
VolSessionId=1
VolSessionTime=1025016612
Volume=File0005
VolSessionId=2
VolSessionTime=1025016612
Volume=File0006
VolSessionId=2
VolSessionTime=1025025494
```



Bacula 1.29 User's Guide Chapter 19

Installing and Configuring MySQL

Installing and Configuring MySQL −− Phase I

If you use the ./configure −−with−mysql=mysql−directory statement for configuring **Bacula**, you will need MySQL version 3.23.33 or later installed in the **mysql−directory** (we are currently using 3.23.40). If MySQL is installed in the standard system location, you need only enter **−−with−mysql** since the configure program will search all the standard locations. If you install MySQL in your home directory or some other non−standard directory, you will need to provide the full path to it.

To minimize the problems you may encounter in finding and installing MySQL, we provide a copy of the MySQL source code that we use in the **depkgs1** package.

Installing and Configuring MySQL is not difficult but can be confusing the first time. As a consequence, below, we list the steps that we used to install it on our machines. Please note that our configuration leaves MySQL without any user passwords. This may be an undesirable situation if you have other users on your system.

1. Download MySQL source code from www.mysql.com/downloads or download our **depkgs−mysql** package with the same release number as **Bacula**.

2. Detar it with something like:

   tar xvfz mysql−filename

   Note, the above command requires GNU tar. If you do not have GNU tar, a command such as:

   zcat mysql−filename | tar xvf −

   will probably accomplish the same thing.

3. cd **mysql−source−directory**

   where you replace **mysql−source−directory** with the directory name where you put the MySQL source code.

4. ./configure −−prefix=mysql−directory

   where you replace **mysql−directory** with the directory name where you want to install mysql. Normally for system wide use this is /usr/local/mysql. In my case, I use ~kern/mysql.

5. make

   This takes a bit of time.

6. make install

This will put all the necessary binaries, libraries and support files into the **mysql−directory** that you specified above.

7. ./scripts/mysql_install_db

   This will create the necessary MySQL databases for controlling user access. Note, this script can also be found in the **bin** directory in the installation directory

At this point, you should return to completing the installation of **Bacula**. Later after **Bacula** is installed, come back to this chapter to complete the installation. Please note, the installation files used in the second phase of the MySQL installation are created during the **Bacula** Installation.

Installing and Configuring MySQL −− Phase II

At this point, you should have built and installed MySQL, or already have a running MySQL, and you should have configured, built and installed **Bacula**. If not, please complete these items before proceeding.

Please note that the **./configure** used to build **Bacula** will need to include **−−with−mysql=mysql−directory**, where **mysql−directory** is the directory name that you specified on the ./configure command for configuring MySQL. This is needed so that **Bacula** can find the necessary include headers and library files for interfacing to MySQL.

Now you will create the **Bacula** MySQL database and the tables that **Bacula** uses.

1. Start **mysql**. You might want to use the **startmysql** script provided in the **Bacula** release.
2. cd <bacula−src>/src/cats

   This directory contains the **Bacula** catalog interface routines.

3. ./grant_mysql_privileges

   This script creates unrestricted access rights for **kern**, **kelvin**, and **bacula**. You may want to modify it to suit your situation. Please note that none of these userids including root are password protected.

4. ./create_mysql_database

   This script creates the MySQL **bacula** database. The databases you create as well as the access databases will be located in <install−dir>/var/ in a subdirectory with the name of the database, where <install−dir> is the directory name that you specified on the **−−prefix** option. This can be important to know if you want to make a special backup of the Bacula database or to check its size.

5. ./make_mysql_tables

This script creates the MySQL tables used by **Bacula**.

To take a closer look at the access privileges that you have setup with the above, you can do:

```
mysql−directory/bin/mysql −u root mysql

select * from user;
```

Re−initializing the Catalog Database

After you have done some initial testing with **Bacula**, you will probably want to re−initialize the catalog database and throw away all the test Jobs that you ran. To do so, you can do the following:

```
cd <bacula−source>/src/cats
./drop_mysql_tables
./make_mysql_tables
```

Please note that all information in the database will be lost and you will be starting from scratch. If you have written on any Volumes, you must write and end of file mark on the volume so that **Bacula** can reuse it. Do so with:

```
(stop Bacula or unmount the drive)
mt −f /dev/nst0 rewind
mt −f /dev/nst0 weof
```

Where you should replace **/dev/nst0** with the appropriate tape drive device name for your machine.

Linking Bacula with MySQL

After configuring Bacula with

./configure −−prefix=<mysql−directory>

where <mysql−directory> is in my case **/home/kern/mysql**, you may have to configure the loader so that it can find the MySQL shared libraries. If you put MySQL in a standard place such as **/usr/lib** or **/usr/local/lib** this will not be necessary, but in my case it is. The description that follows is Linux specific. For other operating systems, please consult your manuals on how to do the same thing:

First edit: **/etc/ld.so.conf** and add a new line to the end of the file with the name of the mysql−directory. In my case, it is:

/home/kern/mysql/lib/mysql

then rebuild the loader's cache with:

/sbin/ldconfig

If you upgrade to a new version of **MySQL**, the shared library names will probably changes, and you must re−run the **/sbin/ldconfig** command so that the runtime loader can find them.

Alternatively, your system my have a loader environment variable that can be set. For example, on a Solaris system where I do not have root permission, I use:

```
LD_LIBRARY_PATH=/home/kern/mysql/lib/mysql
```

  
Restoring Files with a Bootstrap File Index Installing and Configuring SQLite

Bacula 1.29 User's Guide Chapter 20

  
Installing and Configuring MySQL Index Internal Bacula Database

Installing and Configuring SQLite

Installing and Configuring SQLite −− Phase I

If you use the **./configure −−with−sqlite** statement for configuring **Bacula**, you will need SQLite version 2.2.3 or later installed. Our standard location (for the moment) for SQLite is in the dependency package **depkgs/sqlite−2.2.3**. Please note that the version will be updated as new versions are available and tested.

Installing and Configuring is quite easy.

1. Download the Bacula dependency packages
2. Detar it with something like:

   tar xvfz depkgs.tar.gz

   Note, the above command requires GNU tar. If you do not have GNU tar, a command such as:

   zcat depkgs.tar.gz | tar xvf −

   will probably accomplish the same thing.

3. **cd depkgs**

4. **make sqlite**

At this point, you should return to completing the installation of **Bacula**.

Please note that the **./configure** used to build **Bacula** will need to include **−−with−sqlite**.

Installing and Configuring SQLite −− Phase II

This phase is done **after** you have run the **./configure** command to configure **Bacula**. At this point, you can create the SQLite database and tables:

1. cd <bacula−src>/src/cats

   This directory contains the **Bacula** catalog interface routines.

2. ./make_sqlite_tables

This script creates the SQLite database as well as the tables used by **Bacula**. This script will be automatically setup by the **./configure** program to create a database named **bacula.db** in **Bacula's** working directory.

Linking Bacula with SQLite

If you have followed the above steps, this will all happen automatically and the SQLite libraries will be linked into **Bacula**.

Testing SQLite

As of this date (20 March 2002), we have much less "production" experience using SQLite than using MySQL. That said, we should note that SQLite has performed flawlessly for us in all our testing.

Re−initializing the Catalog Database

After you have done some initial testing with **Bacula**, you will probably want to re−initialize the catalog database and throw away all the test Jobs that you ran. To do so, you can do the following:

```
cd <bacula-source>/src/cats
./drop_sqlite_tables
./make_sqlite_tables
```

Please note that all information in the database will be lost and you will be starting from scratch. If you have written on any Volumes, you must write and end of file mark on the volume so that **Bacula** can reuse it. Do so with:

```
(stop Bacula or unmount the drive)
mt −f /dev/nst0 rewind
mt −f /dev/nst0 weof
```

Where you should replace **/dev/nst0** with the appropriate tape drive device name for your machine.

  
Installing and Configuring MySQL Index Internal Bacula Database

  
Installing and Configuring SQLite Index Bacula Licenses

Internal Bacula Database

Internal Bacula Database

The **Bacula** internal database is intended to be used primarily by **Bacula** developers for testing; although SQLite is also a good choice for this. We do not recommend its use in general.

This database is simplistic in that it consists entirely of Bacula's internal structures appended sequentially to a file. Consequently, it is in most cases inappropriate for sites with many clients or systems with large numbers of files, or long−term production environments.

Below, you will find a table comparing the features available with SQLite and MySQL and with the internal Bacula database. At the current time, you cannot dynamically switch from one to the other, but must rebuild the Bacula source code. If you wish to experiment with both, it is possible to build both versions of Bacula and install them into separate directories.

Feature	SQLite or MySQL	Bacula
Job Record	Yes	Yes
Media Record	Yes	Yes
FileName Record	Yes	No
File Record	Yes	No
FileSet Record	Yes	Yes
Pool Record	Yes	Yes
Client Record	Yes	Yes
JobMedia Record	Yes	Yes
List Job Records	Yes	Yes
List Media Records	Yes	Yes
List Pool Records	Yes	Yes
List JobMedia Records	Yes	Yes
Delete Pool Record	Yes	Yes
Delete Media Record	Yes	Yes
Update Pool Record	Yes	Yes
Implement Verify	Yes	No

MD5 Signatures	Yes	No

In addition, since there is no SQL available, the Console commands: **sqlquery**, **query**, **retention**, and any other command that directly uses SQL are not available with the Internal database.

  
Installing and Configuring SQLite Index Bacula Licenses

Bacula 1.29 User's Guide Chapter 22

  
Internal Bacula Database Index GPL

Bacula Copyright, Trademark, and Licenses

There are a number of different licenses that are used in Bacula.

GPL

The vast bulk of the code is released under the GNU General Public License version 2. Most of this code is copyrighted: Copyright (C) 2000−2003 Kern Sibbald and John Walker.

Portions may be copyrighted by other people (ATT, the Free Software Foundation, ...).

LGPL

Some of the Bacula library source code is released under the GNU Lesser General Public License. This permits third parties to use these parts of our code in their proprietary programs to interface to Bacula.

Public Domain

Some of the Bacula code has been released to the public domain. E.g. md5.c, SQLite.

Trademark

Bacula®is a registered trademark of Kern Sibbald and John Walker.

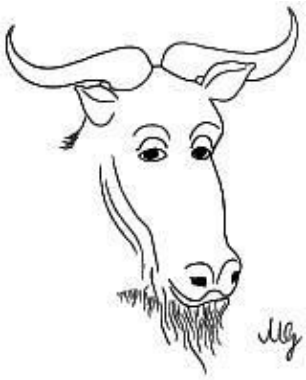
We have trademarked the Bacula name to ensure that any variant of Bacula will be exactly compatible with the program that we have released. The use of the name **Bacula** is restricted to software systems that agree exactly with the program presented here.


Internal Bacula Database


Index


GPL

GNU General Public License



- What to do if you see a possible GPL violation
- Translations of the GPL

Table of Contents

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if

you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- ***a)*** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

- ***b)*** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

- ***c)*** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- ***a)*** Accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

- ***b)*** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine−readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software

interchange; or,

- ***c)*** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty−free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in

other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY

MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.
Copyright (C) *yyyy name of author*

```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA  02111−1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse−clicks or menu items−−whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Return to GNU's home page.

FSF & GNU inquiries & questions to *gnu@gnu.org*. Other ways to contact the FSF.

Comments on these web pages to *webmasters@www.gnu.org*, send other questions to *gnu@gnu.org*.

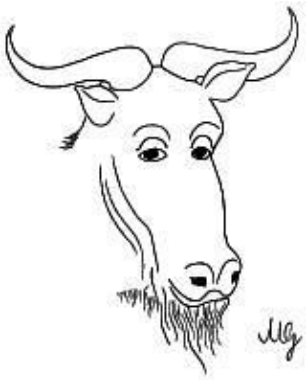
Free Software Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111, USA

Updated: 3 Jan 2000 rms

Bacula Licenses | Index | LGPL

GPL | Index | FAQ

GNU Lesser General Public License



[English | Japanese]

- Why you shouldn't use the Lesser GPL for your next library
- *What to do if you see a possible LGPL violation*
- *Translations of the LGPL*
- The GNU Lesser General Public License as a text file
- The GNU Lesser General Public License as a Texinfo file

This GNU Lesser General Public License counts as the successor of the GNU Library General Public License. For an explanation of why this change was necessary, read the Why you shouldn't use the Lesser GPL for your next library article.

Table of Contents

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

```
Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
 as the successor of the GNU Library Public License, version 2, hence
 the version number 2.1.]
```

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share

and change free software−−to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages−−typically libraries−−of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two−step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non−free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non−free programs. These disadvantages are the reason we

use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de−facto standard. To achieve this, non−free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non−free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non−free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non−free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on

what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- ***a)*** The modified work must itself be a software library.
- ***b)*** You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- ***c)*** You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- ***d)*** If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

  (For example, a function in a library to compute square roots has a purpose that is entirely well−defined independent of the application. Therefore, Subsection 2d requires that any application−supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

  These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

  Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

  In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to

this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- ***a)*** Accompany the work with the complete corresponding machine−readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine−readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- ***b)*** Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface−compatible with the version that the work was made with.
- ***c)*** Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- ***d)*** If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- ***e)*** Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side−by−side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- ***a)*** Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- ***b)*** Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty−free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the library's name and an idea of what it does.
Copyright (C) *year name of author*

```
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in
the library `Frob' (a library for tweaking knobs) written
by James Random Hacker.
```

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

Return to GNU's home page.

FSF & GNU inquiries & questions to *gnu@gnu.org*. Other ways to contact the FSF.

Comments on these web pages to *webmasters@www.gnu.org*, send other questions to *gnu@gnu.org*.

Copyright notice above.
Free Software Foundation, Inc., 59 Temple Place − Suite 330, Boston, MA 02111, USA

Updated: 27 Nov 2000 paulv

END OF TERMS AND CONDITIONS

Bacula Frequently Asked Questions

See the bugs section of this document for a list of known bugs and solutions.

*What is **Bacula**?*

Bacula is a network backup and restore program.

*Does **Bacula** support Windows?*

Yes, **Bacula** compiles and runs on Windows machines (Win95, Win98, WinMe, WinXP, WinNT, and Win2000). We provide a binary version of the Client (bacula−fd), but have not tested the Director nor the Storage daemon.

*What language is **Bacula** written in?*

It is written mostly in C, however, it is completely compiled using the C++ compiler. There are several modules, including the Win32 interface that are written in C++, and over time, we are slowly converting to using a small subset of C++.

*On what machines does **Bacula** run?*

Bacula builds and executes on RedHat Linux (versions 7.1, 7.2, and 7.3), FreeBSD, Solaris, and Win32. On Win32 systems (Win95/98/Me/XP/NT/2000), only the client program (File daemon) has been tested.
Bacula has been my primary (only) backup tool for over two years backing up 5 machines nightly (3 Linux boxes running RedHat, a Win98 machine, and a WinNT machine).

*Is **Bacula** stable?*

Yes, it is remarkably stable, but remember, there are still a lot of unimplemented or partially implemented features. With a program of this size (60,000+ lines of C code not including the SQL programs) there are bound to be bugs. The current test environment (a twisted pair local network and a HP DLT backup tape) is rather idea, so additional testing on other sites is necessary. The File daemon has never crashed −− running months at a time with no intervention. The Storage daemon is remarkably stable with most of the problems arising during labeling or switching tapes. Storage daemon crashes are rare. The Director, given the multitude of functions it fulfills is also relatively stable. In a production environment, it rarely if ever crashes. Of the three daemons, the Director is the most prone to having problems. It frequently runs several months with no problems.

There are a number of reasons for this stability. 1. The program was largely written by one person to date (Kern). 2. The program constantly is checking the chain of allocated memory buffers to ensure that no overruns have occurred. 3. All memory leaks (orphaned buffers) are reported each time the program terminates. 4. Any signal (segmentation fault, ...) generates a traceback that is emailed to the developer. This permits quick resolution of bugs even if they only show up rarely in a production system.

I'm Getting Authorization Errors. What is Going On?

For security reasons, **Bacula** requires that both the File daemon and the Storage daemon know the name of the Director as well as his password. As a consequence, if you change the Director's name or password, you must make the corresponding change in the Storage daemon and in the File daemon configuration files.
During the authorization process, the Storage daemon and File daemon also require that the Director authenticate himself, so both ends require the other to have the correct name and password.

My Catalog is Full of Test Runs, How Can I Start Over?

If you are using MySQL do the following:

```
cd <bacula-source>/src/cats
./drop_mysql_tables
./make_mysql_tables
```

If you are using SQLite, do the following:

```
Delete bacula.db from your working directory.
cd <bacula-source>/src/cats
./drop_sqlite_tables
./make_sqlite_tables
```

Then write an EOF on each tape you used with **Bacula**

I Run a Restore Job and Bacula Hangs. What do I do?

On Bacula version 1.25 and prior, it expects you to have the correct tape mounted prior to a restore. On Bacula version 1.26 and higher, it will ask you for the tape, and if the wrong one it mounted, it will inform you.
If you have previously done an **unmount** command, all Storage daemon sessions (jobs) will be completely blocked from using the drive unmounted, so be sure to do a **mount** after your unmount. If in doubt, do a second **mount**, it won't cause any harm.

I Cannot Get My Windows Client to Start Automatically?

You are probably having one of two problems: either the Client is dying due to an incorrect configuration file, or you didn't do the Installation commands necessary to install it as a Windows Service.
For the first problem, see the next FAQ question. For the second problem, please review the Windows Installation instructions in this manual.

My Windows Client Immediately Dies When I Start It

The most common problem is either that the configuration file is not where it expects it to be, or that there is an error in the configuration file. You must have the configuration file in **c:\bacula\bin\bacula-fd.conf**.
To **see** what is going on when the File daemon starts on Windows, do the following:

```
Start a DOS shell Window.
cd c:\bacula\bin
bacula-fd -t >out
```

```
type out
```

Calling **bacula−fd** with redirection (>) will write the diagnostic output to the file **out** which you can then list. The **−t** option tells **Bacula** to read the configuration file and then exit.

When I Start the Console, the Error Messages Fly By. How can I see them?

Either use a shell window with a scroll bar, or use the gnome−console. In any case, you probably should be logging all output to a file, and then you can simply view the file using an editor or the **less** program. To log all output, I have the following in my Director's Message resource definition:

```
append = "/home/kern/bacula/bin/log" = all, !skipped
```

Obviously you will want to change the filename to be appropriate for your system.

I didn't realize that the backups were not working on my Windows Client. What should I do?

You should be sending yourself an email message for each job. This will avoid the possibility of not knowing about a failed backup. To do so put something like:

```
Mail = yourname@yourdomain = all, !skipped
```

in your Director's message resource. You should then receive one email for each Job that ran. When you are comfortable with what is going on (it took me 9 months), you might change that to:

```
MailOnError = yourname@yourdomain = all, !skipped
```

then you only get email messages when a Job errors as is the case for your Windows machine.

You should also be logging the Director's messages, please see the previous FAQ for how to do so.

All my Jobs are scheduled for the same time. Will this cause problems?

No, not at all. **Bacula** will schedule all the Jobs at the same time, but will run them one after another unless you have increased the number of simultaneous jobs in the configuration files for the Director, the File daemon, and the Storage daemon. The appropriate configuration record is **Maximum Concurrent Jobs = nn**.

Can Bacula Backup My System To Files instead of Tape?

Yes, in principle, **Bacula** can backup to any storage medium as long as you have correctly defined that medium in the Storage daemon's Device resource. For an example of how to backup to files, please see the Pruning Example in the Recycling chapter of this manual.

Can Bacula Backup and Restore Files Greater than 2 Giga Bytes in Size?

If your operating system permits it, and you are running Bacula version 1.26 or later, the answer is yes. Unfortunately, large files are not supported by **cygwin** which Bacula uses for the Windows version client. Just the same, when **Bacula** saves and restores to tape media, it does not use seek() calls, thus there should be no problems with file sizes.

*I Started A Job then Decided I Really Did Not Want to Run It. Is there a better way than **./bacula stop** to stop it?*

Yes, you normally should use the Console command **cancel** to cancel a Job that is either scheduled or running. If the Job is scheduled, it will be marked for cancellation and will be canceled when it is scheduled to start. If it is running, it will normally terminate after a few minutes. If the Job is waiting on a tape mount, you will need to do a **mount** command before it will be canceled.

Why have You Trademarked the Name Bacula®?

We have trademarked the name **Bacula** to ensure that all media written by any program named **Bacula** will always be compatible. Anyone may use the name Bacula, even in a derivative product as long as it remains totally compatible in all respects with the program defined here.

Why is Your Online Document for Version 1.27 of Bacula when the Currently Release Version is 1.26?

As Bacula is being developed, the document is also being enhanced, more often than not it has clarifications of existing features that can be very useful to our users, so we publish the very latest document. Fortunately it is rare that there are confusions with new features.
If you want to read a document that pertains only to a specific version, please use the one distributed in the source code.

How Can You Be Sure that Bacula Really Saves and Restores All Files?

It is really quite simple, but took me awhile to figure out how to "prove" it. First make a Bacula Rescue disk, see the Disaster Recovery Using Bacula of this manual. Second, you run a full backup of all your files on all partitions. Third, you run an Verify InitCatalog Job on the same FileSet, which effectively makes a record of all the files on your system. Fourth, you run a Verify Catalog job and assure yourself that nothing has changed (well, between an InitCatalog and Catalog one doesn't expect anything). Then do the unthinkable, write zeros on your MBR (master boot record) wiping out your hard disk. Now, restore your whole system using your Bacula Rescue disk and the Full backup you made, and finally re−run the Verify Catalog job. You will see that with the exception of the directory modification and access dates and the files changed during the boot, your system is identical to what it was before you wiped your hard disk.

How Can You Claim to Handle Unlimited Path and Filename Lengths when All Other Programs Have Fixed Limits?

Most of those other programs have been around for a long time, in fact since the beginning of Unix, which means that they were designed for rather small fixed length path and filename lengths. Over the years, these restrictions have been relaxed allowing

longer names. Bacula on the other hand was designed in 2000, and so from the start, Path and Filenames have been keep in buffers that start at 256 bytes in length but can grow as needed to handle any length. Most of the work is carried out by lower level routines making the coding rather easy.

What Is the Really Unique Feature of Bacula?

Well, it is hard to come up with unique features when backup programs for Unix machines have been around since the 1960s. That said, I believe that Bacula is the first and only program to use a standard SQL interface to its catalog database. Although this adds a bit of complexity and overhead, it provides an amazingly rich set of features that are easy to program and enhance. The current code has barely scratched the surface in this regard.
The second unique feature, which is currently (1.28) unimplemented, and thus can be called vaporware :−), is Base level saves. When implemented, this will enormously reduce tape usage.

Since Bacula is Multithreaded, Why Do You Recommend Not Running Multiple Simultaneous Jobs?

Bacula can run multiple simultaneous jobs, but being very conservative, I do not recommend doing so at the current time for two reasons: 1. I'm not sure that two jobs accessing the same Volume won't have catalog conflicts. 2. Doing a restore from a Volume with records from two jobs intermingled is complicated.
I expect to have definitive solutions to both of those problems in a later version. In the mean time, I recommend running one Job at a time. If you want to experiment with this as several large shops are doing, ensure that each job is writing to a different Volume (i.e. has a different Storage resource). This will avoid the two problems mentioned above. Just the same, there may be other race conditions in the Director that cause problems. Be forewarned.

If I Do Run Multiple Simultaneous Jobs, How Can I Force One Particular Job to Run After Another Job?

This is possible by using the **RunBeforeJob** and **RunAfterJob** statements. For example, assume that you want job B to start only after job A has completely finished. First schedule Job A to start at least one minute before job B. Then within Job A, use the RunBeforeJob to create a "lock" file and the RunAfterJob to delete that file, and within Job B, make the RunBeforeJob statement call a small shell script that waits until the "lock" file does not exist. For example: Job A:

```
RunBeforeJob = "touch /tmp/Bacula−run.lock"
RunAfterJob = "rm −f /tmp/Bacula−run.lock"
```

and in Job B use:

```
RunBeforeJob = "while true ; do if [ −f /tmp/Bacula−run.lock ] ;
                then sleep 5; else break; fi; done"
```

where the above RunBeforeJob is all entered on a single line. With the above statements, Job B will start running just after Job A runs, but it will be blocked until a few seconds (0 − 5) after Job

A terminates.


LGPL

Index

Projects


FAQ

Index

Security Issues

Bacula Projects

Projects

- Test as many features of **Bacula** as possible. This is happening a bit with every release.
- Ensure that the documentation is complete and clear.
- Implement the Bacula Roadmap as defined below.
- Enhance the file restore code to restore according to a large number of criteria (include list, exclude list, state of system as of date xxx, ...) (assigned to Kern)
- Write a GUI interface for restoring files.
- Define and write a regression script.
- Implement verification of tape data −− compared to what is on disk.
- Implement SSL between daemons.
- Performance tuning.
- Implement a scripting language for running Jobs.
- Implement GNOME Bacula applet.
- Implement new serial code for daemon−daemon comm and replace all sscanf's.
- Complete review of SQL database usage, indexes, and table design.

Raw Todo List

Please see the file **kernstodo** in the main **Bacula** directory for the raw todo list.

Bacula Projects Roadmap

```
            Bacula Projects Roadmap
               17 August 2002
            last update 5 January 2003

Item 1:   Multiple simultaneous Jobs. (done)
Done −− Restore part needs better implementation to work correctly

  What:   Permit multiple simultaneous jobs in Bacula.

  Why:    An enterprise level solution needs to go fast without the
          need for the system administrator to carefully tweak
          timing.  Based on the benchmarks, during a full
          backup, NetWorker typically hit 10 times the bandwidth to
          the tape compared to Bacula−−largely. This is probably due to
          running parallel jobs and multi−threaded filling of buffers
          and writing them to tape.  This should also make things work
          better when you have a mix of fast and slow machines backing
          up at the same time.

  Notes:  Bacula was designed to run multiple simultaneous jobs. Thus
          implementing this is a matter of some small cleanups and
          careful testing.


Item 2:   Make the Storage daemon use intermediate file storage to buffer data.
Deferred −− not necessary yet.

  What:   If data is coming into the SD too fast, buffer it to
```

disk if the user has configured this option.

Why: This would be nice, especially if it more or less falls out when implementing (1) above. If not, it probably should not be given a high priority because fundamentally the backup time is limited by the tape bandwidth. Even though you may finish a client job quicker by spilling to disk, you still have to eventually get it onto tape. If intermediate disk buffering allows us to improve write bandwidth to tape, it may make sense.

Notes: Whether or not this is implemented will depend upon performance testing after item 1 is implemented.

Item 3: Write the bscan program -- also write a bcopy program.
Done

What: Write a program that reads a Bacula tape and puts all the appropriate data into the catalog. This allows recovery from a tape that is no longer in the database, or it allows re-creation of a database if lost.

Why: This is a fundamental robustness and disaster recovery tool which will increase the comfort level of a sysadmin considering adopting Bacula.

Notes: A skeleton of this program already exists, but much work needs to be done. Implementing this will also make apparent any deficiencies in the current Bacula tape format.

Item 4: Implement Base jobs.

What: A base job is sort of like a Full save except that you will want the FileSet to contain only files that are unlikely to change in the future (i.e. a snapshot of most of your system after installing it). After the base job has been run, when you are doing a Full save, you can specify to exclude all files saved by the base job that have not been modified.

Why: This is something none of the competition does, as far as we know (except BackupPC, which is a Perl program that saves to disk only). It is big win for the user, it makes Bacula stand out as offering a unique optimization that immediately saves time and money.

Notes: Big savings in tape usage. Will require more resources because the e. DIR must send FD a list of files/attribs, and the FD must search the list and compare it for each file to be saved.

Item 5: Implement Label templates

What: This is a mechanism whereby Bacula can automatically create a tape label for new tapes according to a detailed specification provided by the user.

Why: It is a major convenience item for folks who use automated label creation.

Notes: Bacula already has a working form of automatic tape label creation, but it is very crude. The design for the complete tape labeling project is already documented in the manual.

Item 6: Write a regression script.
Started

What: This is an automatic script that runs and tests as many features of Bacula as possible. The output is compared to previous versions of Bacula and any differences are reported.

Why: This is an enormous help in preventing introduction of new errors in parts of the program that already work correctly.

Notes: This probably should be ranked higher, it's something the typical user doesn't see. Depending on how it's implemented, it may make sense to defer it until the archival tape format and user interface mature.

Item 7: GUI for interactive restore
Item 8: GUI for interactive backup

What: The current interactive restore is implemented with a tty interface. It would be much nicer to be able to "see" the list of files backed up in typical GUI tree format. The same mechanism could also be used for creating ad-hoc backup FileSets (item 8).

Why: Ease of use -- especially for the end user.

Notes: Rather than implementing in Gtk, we probably should go directly for a Browser implementation, even if doing so meant the capability wouldn't be available until much later. Not only is there the question of Windows sites, most Solaris/HP/IRIX, etc, shops can't currently run Gtk programs without installing lots of stuff admins are very wary about. Real sysadmins will always use the command line anyway, and the user who's doing an interactive restore or backup of his own files will in most cases be on a Windows machine running Exploder.

Item 9: Add SSL to daemon communications.

What: This provides for secure communications between the daemons.

Why: This would allow doing backup across the Internet without privacy concerns (or with much less concern).

Notes: The vast majority of near term potential users will be backing up a single site over a LAN and, correctly or not, they probably won't be concerned with security, at least not enough to go to the trouble to set up keys, etc. to screw things down. We suspect that many users genuinely interested in multi-site backup already run some form of VPN software in their internetwork connections, and are willing to delegate security to that layer.

Item 10: Define definitive tape format.

```
Done (version 1.27)

  What:   Define that definitive tape format that will not change
          for the next millennium.

  Why:    Stability, security.

  Notes:  See notes for item 11 below.


Item 11:  New daemon communication protocol.

  What:   The current daemon to daemon protocol is basically an ASCII
          printf() and sending the buffer. On the receiving end, the
          buffer is sscanf()ed to unpack it. The new scheme would
          be a binary format that allows quick packing and unpacking
          of any data type with named fields.

  Why:    Using binary packing would be faster. Named fields will permit
          error checking to ensure that what is sent is what the
          receiver really wants.

  Notes:  These are internal improvements in the interest of the
          long-term stability and evolution of the program.  On the one
          hand, the sooner they're done, the less code we have to rip
          up when the time comes to install them.  On the other hand, they
          don't bring an immediately perceptible benefit to potential
          users.  Item 10 and possibly item 11 should be deferred until Bacula
          is well established with a growing user community more or
          less happy with the feature set.  At that time, it will make a
          good "next generation" upgrade in the interest of data
          immortality.
```

FAQ Index Security Issues

FAQ Index Thanks

Bacula Security Issues

- 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.
- 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.
- 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).
- Currently all data that is sent across the network is unencrypted. As a consequence, unless you use **ssh** for port forwarding, it is not recommended to do a backup across an insecure network (e.g. the Internet). In a future version, we plan to have **ssh** encryption built−in.
- 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.
- 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.

TCP Wrappers

TCP Wrappers are implemented if you turn them on when configuring (**./configure −−with−libwrap**). With this code enabled, you may control who may access your daemons. This control is done by modifying the file: **/etc/hosts.allow**. This code is implemented but untested.


FAQ


Index


Thanks

Bacula 1.29 User's Guide Chapter


Security Issues


Index


Bugs

Thanks

Thanks to Richard Stallman for starting the Free Software movement and for bringing us gcc and all the other GNU tools.

Thanks to Linus Torvolds for bring us Linux.

Thanks to all the Free Software programmers. Without being able to peek at your code, and in some cases, take parts of it, this project would have been much more difficult.

Thanks to John Walker for suggesting this project, giving it a name, contributing software he has written, and for his programming efforts on Bacula as well as having acted as a constant sounding board and source of ideas.

Thanks to the apcupsd project where I started my Free Software efforts, and from which I was able to borrow some ideas and code that I had written.

For all those who I have left out, thanks.

Copyrights and Trademarks

Certain words and/or products are Copyrighted or Trademarked such as Windows (by Microsoft). Since they are numerous, and we are not necessarily aware of the details of each, we don't try to list them here. However, we acknowledge all such Copyrights and Trademarks, and if any copyright or trademark holder wishes a specific acknowledgment, notify us, and we will be happy to add it.


Security Issues

Index

Bugs


Thanks

Index

Bacula Director Services

Bacula Bugs

Well there certainly are bugs, and one day when additional man/woman power exists on the **Bacula** project, they will be reported here, but for the moment, all we can say is that **Bacula** is amazing robust considering the amount of code and complexity of its tasks.

A "raw" list of the current known issues can be found in **kernstodo** in the main Bacula source directory.

Director Services Daemon

General

This chapter is intended to be a technical discussion of the Catalog services and as such is not targeted at end users but rather at developers and system administrators that want or need to know more of the working details of **Bacula**.

The **Bacula Director** services consist of the program that supervises all the backup and restore operations. The system administrator uses the Bacula Console to communicate to the Director daemon in order to schedule backups and to recover files. In general, the Director can perform nine major tasks (only three of which are currently implemented):

Backup
: Backup a FileSet to Backup volumes. File and Backup Volume information is stored in the Catalog.

Restore
: Restore files from Backup Volumes.

Verify
: Verify that a FileSet corresponds to what is in the Catalog or what is on the Backup Volumes. Note that not all options of Verify are currently implemented. This option makes a quite reasonable **Tripwire** substitute.

**Archive*
: Backup a FileSet to Archived volumes and optionally remove the files from the system.

**Clone*
: Make a copy of the FileSet on Cloned volumes and put the volume information in the Catalog database.

**Copy*
: Make a copy of the FileSet on Copy volumes. No information on Copy volumes is stored in the Catalog database.

**Save*
: Save a FileSet to Save Volumes. No information on Save volumes is stored in the Catalog database.

**Migration*
: Migrate the specified FileSet to different media.

**Scan*
: Restore information on the Volumes scanned to the Catalog database.

**Update*
: Ensure that the files on a remote file system are updated to the specified FileSet (a sort of RDIST functionality).

Storage Daemon Design

General

This chapter is intended to be a technical discussion of the Storage daemon services and as such is not targeted at end users but rather at developers and system administrators that want or need to know more of the working details of **Bacula**.

Introduction

The Bacula Storage daemon provides storage resources to a Bacula installation. An individual Storage daemon is associated with a physical permanent storage device (for example, a tape drive, CD writer, tape changer or jukebox, etc.), and may employ auxiliary storage resources (such as space on a hard disk file system) to increase performance and/or optimize use of the permanent storage medium.

Any number of storage daemons may be run on a given machine; each associated with an individual storage device connected to it, and BACULA operations may employ storage daemons on any number of hosts connected by a network, local or remote. The ability to employ remote storage daemons (with appropriate security measures) permits automatic off−site backup, possibly to publicly available backup repositories.

Development Outline

In order to provide a high performance backup and restore solution that scales to very large capacity devices and networks, the storage daemon must be able to extract as much performance from the storage device and network with which it interacts. In order to accomplish this, storage daemons will eventually have to sacrifice simplicity and painless portability in favor of techniques which improve performance. My goal in designing the storage daemon protocol and developing the initial prototype storage daemon is to provide for these additions in the future, while implementing an initial storage daemon which is very simple and portable to almost any POSIX−like environment. This original storage daemon (and its evolved descendants) can serve as a portable solution for non−demanding backup requirements (such as single servers of modest size, individual machines, or small local networks), while serving as the starting point for development of higher performance configurable derivatives which use techniques such as POSIX threads, shared memory, asynchronous I/O, buffering to high−speed intermediate media, and support for tape changers and jukeboxes.

Connections and Sessions

A client connects to a storage server by initiating a conventional TCP connection. The storage server accepts the connection unless its maximum number of connections has been reached or the specified host is not granted access to the storage server. Once a connection has been opened, the client may make any number of Query requests, and/or initiate (if permitted), one or more Append sessions (which transmit data to be stored by the storage daemon) and/or Read sessions (which retrieve data from the storage daemon).

Most requests and replies sent across the connection are simple ASCII strings, with status replies

prefixed by a four digit status code for easier parsing. Binary data appear in blocks stored and retrieved from the storage. Any request may result in a single−line status reply of "`3201 Notification pending`", which indicates the client must send a "Query notification" request to retrieve one or more notifications posted to it. Once the notifications have been returned, the client may then resubmit the request which resulted in the 3201 status.

The following descriptions omit common error codes, yet to be defined, which can occur from most or many requests due to events like media errors, restarting of the storage daemon, etc. These details will be filled in, along with a comprehensive list of status codes along with which requests can produce them in an update to this document.

Append Requests

append open session = <JobId> [<Password>]

A data append session is opened with the Job ID given by *JobId* with client password (if required) given by *Password*. If the session is successfully opened, a status of `3000 OK` is returned with a "`ticket =` *number*" reply used to identify subsequent messages in the session. If too many sessions are open, or a conflicting session (for example, a read in progress when simultaneous read and append sessions are not permitted), a status of "`3502 Volume busy`" is returned. If no volume is mounted, or the volume mounted cannot be appended to, a status of "`3503 Volume not mounted`" is returned.

append data = <ticket−number>

If the append data is accepted, a status of `3000 OK data address = <IPaddress> port = <port>` is returned, where the `IPaddress` and `port` specify the IP address and port number of the data channel. Error status codes are `3504 Invalid ticket number` and `3505 Session aborted`, the latter of which indicates the entire append session has failed due to a daemon or media error. Once the File daemon has established the connection to the data channel opened by the Storage daemon, it will transfer a header packet followed by any number of data packets. The header packet is of the form:

```
<file−index> <stream−id> <info>
```

The details are specified in the Daemon Protocol section of this document.

**append abort session = <ticket−number>*

The open append session with ticket *ticket−number* is aborted; any blocks not yet written to permanent media are discarded. Subsequent attempts to append data to the session will receive an error status of `3505 Session aborted`.

append end session = <ticket−number>

The open append session with ticket *ticket−number* is marked complete; no further blocks may be appended. The storage daemon will give priority to saving any buffered blocks from this session to permanent media as soon as possible.

append close session = <ticket−number>

The append session with ticket *ticket* is closed. This message does not receive an `3000 OK` reply until all of the content of the session are stored on permanent media, at which time said reply is given, followed by a list of volumes, from first to last, which contain blocks from the session, along with the first and last file and block on each

containing session data and the volume session key identifying data from that session in lines with the following format:
`Volume = <Volume-id> <start-file> <start-block> <end-file> <end-block> <volume-session-id>`where *Volume-id* is the volume label, *start-file* and *start-block* are the file and block containing the first data from that session on the volume, *end-file* and *end-block* are the file and block with the last data from the session on the volume and *volume-session-id* is the volume session ID for blocks from the session stored on that volume.

Read Requests

Read open session = <JobId> <Volume-id> <start-file> <start-block> <end-file> <end-block> <volume-session-id> <password>

where *Volume-id* is the volume label, *start-file* and *start-block* are the file and block containing the first data from that session on the volume, *end-file* and *end-block* are the file and block with the last data from the session on the volume and *volume-session-id* is the volume session ID for blocks from the session stored on that volume.
If the session is successfully opened, a status of

`3100 OK Ticket = number"`

is returned with a reply used to identify subsequent messages in the session. If too many sessions are open, or a conflicting session (for example, an append in progress when simultaneous read and append sessions are not permitted), a status of "`3502 Volume busy`" is returned. If no volume is mounted, or the volume mounted cannot be appended to, a status of "`3503 Volume not mounted`" is returned. If no block with the given volume session ID and the correct client ID number appears in the given first file and block for the volume, a status of "`3505 Session not found`" is returned.

Read data = <Ticket> > <Block>

The specified Block of data from open read session with the specified Ticket number is returned, with a status of `3000 OK` followed by a "`Length =` *size*" line giving the length in bytes of the block data which immediately follows. Blocks must be retrieved in ascending order, but blocks may be skipped. If a block number greater than the largest stored on the volume is requested, a status of "`3201 End of volume`" is returned. If a block number greater than the largest in the file is requested, a status of "`3401 End of file`" is returned.

Read close session = <Ticket>

The read session with Ticket number is closed. A read session may be closed at any time; you needn't read all its blocks before closing it.

---

by John Walker
January 30th, MM

File Services Daemon

General

Please note, this section is somewhat out of date as the code has evolved significantly. The basic idea has not changed though.

This chapter is intended to be a technical discussion of the File daemon services and as such is not targeted at end users but rather at developers and system administrators that want or need to know more of the working details of **Bacula**.

The **Bacula File Services** consist of the programs that run on the system to be backed up and provide the interface between the Host File system and Bacula −− in particular, the Director and the Storage services.

When time comes for a backup, the Director gets in touch with the File daemon on the client machine and hands it a set of "marching orders" which, if written in English, might be something like the following:

OK, **File daemon**, it's time for your daily incremental backup. I want you to get in touch with the Storage daemon on host archive.mysite.com and perform the following save operations with the designated options. You'll note that I've attached include and exclude lists and patterns you should apply when backing up the file system. As this is an incremental backup, you should save only files modified since the time you started your last backup which, as you may recall, was 2000−11−19−06:43:38. Please let me know when you're done and how it went. Thank you.

So, having been handed everything it needs to decide what to dump and where to store it, the File daemon doesn't need to have any further contact with the Director until the backup is complete providing there are no errors. If there are errors, the error messages will be delivered immediately to the Director. While the backup is proceeding, the File daemon will send the file coordinates and data for each file being backed up to the Storage daemon, which will in turn pass the file coordinates to the Director to put in the catalog.

During a **Verify** of the catalog, the situation is different, since the File daemon will have an exchange with the Director for each file, and will not contact the Storage daemon.

A **Restore** operation will be very similar to the **Backup** except that during the **Restore** the Storage daemon will not send storage coordinates to the Director since the Director presumably already has them. On the other hand, any error messages from either the Storage daemon or File daemon will normally be sent directly to the Directory (this, of course, depends on how the Message resource is defined).

Commands Received from the Director for a Backup

type = <type>
: The backup type specifies the type of backup to be done. It may be one of the following: full, incremental, differential, level N, since date

  where

full
is all files in the FileSet whether or not they have changed.
incremental
is all files that have changed since the last backup.
differential
is all files that have changed since the last Full backup.
**level N*
is all files that have changed since the last Level N backup.
**since date*
is all files that have changed since the date specified.

jobid = <number>
This is the unique JobId of the current Job. It is saved on the storage media with the File Attributes.

storage address = <Storage−daemon−hostname>port = <port−number>
The Storage Address specifies the host name of the Storage daemon as well as its port number.

include = <processing−attribute−list>
<file−list>
The Include Files specifies the list of files and/or directories to be included in the backup job. The <processing−attribute−list> is optional. If specified, it is a comma separated list of processing options to be applied to the file−list. These options are used to modify the default processing behavior of the files included. The options to be modified may be one of the following:

compression=
**GZIP*
All files saved will be compressed using the GNU ZIP compression format. This code is only partially implemented please DO NOT USE it.

signature=
MD5
An MD5 signature will be computed for all files saved.

encryption=
NOT IMPLEMENTED. PLEASE DO NOT USE.
**blowfish*
Blowfish encryption will be applied to all files saved.
**3DES*
The 3DES encryption will be applied to all files saved.

verify=
Report changes in any of the following:
p
permission bits
i
inode number
n
number of links
u
userid
g
groupid
s
file size change
a

access time/date

m
: modification time/date

c
: CTIME ref time/date

5
: MD5 signature

d
: file size decrease

Miscellaneous not yet implemented. PLEASE DO NOT USE.

noexec
: All executable files will be automatically excluded from being saved.

nofollow
: Any symbolic link found will be saved, but the link will not be followed. Thus the file to which the link points may not be saved.

nonfs
: Any file that resides on an NFS filesystem will not be saved.

nodescent
: Do not descend into directories found.

sparse
: Enable special code that checks for sparse files such as created by ndbm.

onefs
: Remain on a single filesystem.

The **<file−list>** is a list of filenames and/or directory names with each name being on a separate line. The filename may be enclosed in quotes for easy inclusion of spaces.

exclude
<file−list>
: The Exclude Files specifies the list of files and/or directories to be excluded from the backup job. The <file−list> is a list of filenames and/or directory names with each name being on a separate line. The <file−list> may contain wild−cards.

save
: The save command causes the **File daemon** to begin sending the listed files to the Storage daemon to be saved.

Commands Received from the Director for a Restore

To be written ...

Catalog Services

General

This chapter is intended to be a technical discussion of the Catalog services and as such is not targeted at end users but rather at developers and system administrators that want or need to know more of the working details of **Bacula**.

The **Bacula Catalog** services consist of the programs that provide the SQL database engine for storage and retrieval of all information concerning files that were backed up and their locations on the storage media.

We have investigated the possibility of using the following SQL engines for Bacula: Beagle, mSQL, GNU SQL, PostgreSQL, and MySQL. Each presents certain problems with either licensing or maturity. At present, we have chosen for development purposes to use MySQL and SQLite. MySQL was chosen because it is fast, proven to be reliable, widely used, and actively being developed. In addition, the current version of MySQL is released under the GNU GPL license. SQLite was chosen because it is small, efficient, and can be directly embedded in **Bacula** thus requiring much less effort from the system administrator or person building **Bacula**. In our testing SQLite has performed very well, and for the functions that we use, it has never encountered any errors. Our experience using SQLite is, however, somewhat limited at this time (24 March 2002) compared to MySQL.

The **Bacula** SQL code has been written in a manner that will allow it to be easily modified to support any of the current SQL database systems on the market (for example: PostgreSQL, mSQL, iODBC, unixODBC, Solid, OpenLink ODBC, EasySoft ODBC, InterBase, Oracle8, Oracle7, and DB2).

If you do not specify either **−−with−mysql** or **−−with−sqlite** on the ./configure line, **Bacula** will use its minimalist internal database. This is not recommended except for some developers. At the current time, this internal database supports most of the features needed by Bacula (job records, pool records, media records, ...). However, it does not currently keep track of the filenames saved for each job nor permit any SQL command. In addition, there is no means for doing a **retention** command on the internal database (i.e. no way to trim the growing size of the database).

Filenames and Maximum Filename Length

On Kern's Linux RedHat 7.1 development system, there are approximately 90,000 files, of which 149 have a filename length (including full path) greater than 100 characters. The maximum filename length found on this system is 133 bytes.

In general, either MySQL or SQLite permit storing arbitrary long path names and file names in the catalog database. In practice, there are still one or two places in the Catalog interface code that restrict the maximum path length to 512 characters and the maximum file name length to 512 characters. These restrictions will be removed in the next major release. Please note, these restrictions apply only to the Catalog database and thus to your ability to list online the files saved during any job. All information received and stored by the Storage daemon (normally on tape) allows and handles arbitrarily long path and filenames.

Installing and Configuring MySQL

For the details of installing and configuring MySQL, please see the Installing and Configuring MySQL chapter of this manual.

Installing and Configuring SQLite

For the details of installing and configuring SQLite, please see the Installing and Configuring SQLite chapter of this manual.

Internal Bacula Catalog

Please see the Internal Bacula Database chapter of this manual for more details.

Database Table Design

All discussions that follow pertain to the MySQL database. The details for the SQLite database are essentially identical except for that all fields in the SQLite database are stored as ASCII text and some of the database creation statements are a bit different. The details of the internal Bacula catalog are not discussed here.

Because the Catalog database may contain very large amounts of data for large sites, we have made a modest attempt to normalize the data tables to reduce redundant information. While reducing the size of the database significantly, it does, unfortunately, add some complications to the structures.

In simple terms, the Catalog database must contain a record of all Jobs run by Bacula, and for each Job, it must maintain a list of all files saved, with their File Attributes (permissions, create date, ...), and the location and Media on which the file is stored. This is seemingly a simple task, but it represents a huge amount interlinked data. Note: the list of files and their attributes is not maintained when using the internal Bacula database. The data stored in the File records, which allows the user or administrator to obtain a list of all files backed up during a job, is by far the largest volume of information put into the Catalog database.

Although the Catalog database has been designed to handle backup data for multiple clients, some users may want to maintain multiple databases, one for each machine to be backed up. This reduces the risk of confusion of accidental restoring a file to the wrong machine as well as reducing the amount of data in a single database, thus increasing efficiency and reducing the impact of a lost or damaged database.

Sequence of Creation of Records for a Save Job

Start with StartDate, ClientName, Filename, Path, Attributes, MediaName, MediaCoordinates. (PartNumber, NumParts). In the steps below, "Create new" means to create a new record whether or not it is unique. "Create unique" means each record in the database should be unique. Thus, one must first search to see if the record exists, and only if not should a new one be created, otherwise the existing RecordId should be used.

1. Create new Job record with StartDate; save JobId

2. Create unique Media record; save MediaId
3. Create unique Client record; save ClientId
4. Create unique Filename record; save FilenameId
5. Create unique Path record; save PathId
6. Create unique Attribute record; save AttributeId
   store ClientId, FilenameId, PathId, and Attributes
7. Create new File record
   store JobId, AttributeId, MediaCoordinates, etc
8. Repeat steps 4 through 8 for each file
9. Create a JobMedia record; save MediaId
10. Update Job record filling in EndDate and other Job statistics

Database Tables

Filename		
Column Name	**Data Type**	**Remark**
FilenameId	integer	Primary Key
Name	Blob	Filename

The **Filename** table shown above contains the name of each file backed up with the path removed. If different directories or machines contain the same filename, only one copy will be saved in this table.

Path		
Column Name	**Data Type**	**Remark**
PathId	integer	Primary Key
Path	Blob	Full Path

The **Path** table contains shown above the path or directory names of all directories on the system or systems. The filename and any MSDOS disk name are stripped off. As with the filename, only one copy of each directory name is kept regardless of how many machines or drives have the same directory. These path names should be stored in Unix path name format.

Some simple testing on a Linux file system indicates that separating the filename and the path may be more complication than is warranted by the space savings. For example, this system has a total of 89,097 files, 60,467 of which have unique filenames, and there are 4,374 unique paths.

Finding all those files and doing two stats() per file takes an average wall clock time of 1 min 35 seconds on a 400MHz machine running RedHat 6.1 Linux.

Finding all those files and putting them directly into a MySQL database with the path and filename defined as TEXT, which is variable length up to 65,535 characters takes 19 mins 31 seconds and creates a 27.6 MByte database.

Doing the same thing, but inserting them into Blob fields with the filename indexed on the first 30 characters and the path name indexed on the 255 (max) characters takes 5 mins 18 seconds and creates a 5.24 MB database. Rerunning the job (with the database already created) takes about 2 mins 50 seconds.

Running the same as the last one (Path and Filename Blob), but Filename indexed on the first 30 characters and the Path on the first 50 characters (linear search done there after) takes 5 mins on the average and creates a 3.4 MB database. Rerunning with the data already in the DB takes 3 mins 35 seconds.

Finally, saving only the full path name rather than splitting the path and the file, and indexing it on the first 50 characters takes 6 mins 43 seconds and creates a 7.35 MB database.

File		
Column Name	**Data Type**	**Remark**
FileId	bigint	Primary Key
FileIndex	integer	The sequential file number in the Job
JobId	integer	Link to Job Record
PathId	integer	Link to Path Record
FilenameId	integer	Link to Filename Record
MarkId	integer	Used to mark files during Verify Jobs
LStat	tinyblob	File attributes in base64 encoding
MD5	tinyblob	MD5 signature in base64 encoding

The **File** table shown above contains one entry for each file backed up by Bacula. Thus a file that is backed up multiple times (as is normal) will have multiple entries in the File table. This will probably be the table with the most number of records. Consequently, it is essential to keep the size of this record to an absolute minimum. At the same time, this table must contain all the information (or pointers to the information) about the file and where it is backed up. Since a file may be backed up many times without having changed, the path and filename are stored in separate tables.

This table contains by far the largest amount of information in the Catalog database, both from the stand point of number of records, and the stand point of total database size. As a consequence, the user must take care to periodically reduce the number of File records using the **retention** command in the Console program.

Job

Column Name	Data Type	Remark
JobId	integer	Primary Key
Job	tinyblob	Unique Job Name
Name	tinyblob	Job Name
PurgedFiles	tinyint	Used by Bacula for purging/retention periods
Type	char	Job Type: Backup, Copy, Clone, Archive, Migration
Level	char	Job Level
ClientId	integer	Client index
JobStatus	char	Job Termination Status
SchedTime	datetime	Time/date when Job scheduled
StartTime	datetime	Time/date when Job started
EndTime	datetime	Time/date when Job ended
JobTDate	bigint	Start day in Unix format but 64 bits; used for Retention period.
VolSessionId	integer	Unique Volume Session ID
VolSessionTime	integer	Unique Volume Session Time
JobFiles	integer	Number of files saved in Job
JobBytes	bigint	Number of bytes saved in Job
JobErrors	integer	Number of errors during Job
JobMissingFiles	integer	Number of files not saved (not yet used)
PoolId	integer	Link to Pool Record
FileSetId	integer	Link to FileSet Record

The **Job** table contains one record for each Job run by Bacula. Thus normally, there will be one per day per machine added to the database. Note, the JobId is used to index Job records in the database, and it often is shown to the user in the Console program. However, care must be taken with its use as it is not unique from database to database. For example, the user may have a database for Client data saved on machine Rufus and another database for Client data saved on machine Roxie. In this case, the two database will each have JobIds that match those in another database. For a unique reference to a Job, see Job below.

The Name field of the Job record corresponds to the Name resource record given in the Director's configuration file. Thus it is a generic name, and it will be normal to find many Jobs (or even all

Jobs) with the same Name.

The Job field contains a combination of the Name and the schedule time of the Job by the Director. Thus for a given Director, even with multiple Catalog databases, the Job will contain a unique name that represents the Job.

For a given Storage daemon, the VolSessionId and VolSessionTime form a unique identification of the Job. This will be the case even if multiple Directors are using the same Storage daemon.

FileSet		
Column Name	**Data Type**	**Remark**
FileSetId	integer	Primary Key
FileSet	tinyblob	FileSet name
MD5	tinyblob	MD5 checksum of FileSet

The **FileSet** table contains one entry for each FileSet that is used. The MD5 signature is kept to ensure that if the user changes anything inside the FileSet, it will be detected and the new FileSet will be used. This is particularly important when doing an incremental update. If the user deletes a file or adds a file, we need to ensure that a Full backup is done prior to the next incremental.

JobMedia		
Column Name	**Data Type**	**Remark**
JobMediaId	integer	Primary Key
JobId	integer	Link to Job Record
MediaId	integer	Link to Media Record
FirstIndex	integer	The index (sequence number) of the first file written for this Job to the Media
LastIndex	integer	The index of the last file written for this Job to the Media
StartFile	integer	The physical media (tape) file number of the first block written for this Job
EndFile	integer	The physical media (tape) file number of the last block written for this Job
StartBlock	integer	The number of the first block written for this Job

EndBlock	integer	The number of the last block written for this Job

The **JobMedia** table contains one entry for each volume written for the current Job. If the Job spans 3 tapes, there will be three JobMedia records, each containing the information to find all the files for the given JobId on the tape.

Media		
Column Name	**Data Type**	**Remark**
MediaId	integer	Primary Key
VolumeName	tinyblob	Volume name
Slot	integer	Autochanger Slot number or zero
PoolId	integer	Link to Pool Record
FirstWritten	datetime	Time/date when first written
LastWritten	datetime	Time/date when last written
LabelDate	datetime	Time/date when tape labeled
VolJobs	integer	Number of jobs written to this media
VolFiles	integer	Number of files written to this media
VolBlocks	integer	Number of blocks written to this media
VolMounts	integer	Number of time media mounted
VolBytes	bigint	Number of bytes saved in Job
VolErrors	integer	Number of errors during Job
VolWrites	integer	Number of writes to media
VolMaxBytes	bigint	Maximum bytes to put on this media
VolCapacityBytes	bigint	Capacity estimate for this volume
VolStatus	enum	Status of media: Full, Archive, Append, Recycle, Read−Only, Disabled, Error, Busy
Recycle	tinyint	Whether or not Bacula can recycle the Volumes: Yes, No
VolRetention	bigint	64 bit seconds until expiration

The **Volume** table (internally referred to as the Media table) contains one entry for each volume,

that is each tape, cassette (8mm, DLT, DAT, ...), or file on which information is or was backed up. There is one Volume record created for each of the NumVols specified in the Pool resource record.

Pool		
Column Name	**Data Type**	**Remark**
PoolId	integer	Primary Key
Name	Tinyblob	Pool Name
NumVols	Integer	Number of Volumes in the Pool
MaxVols	Integer	Maximum Volumes in the Pool
UseOnce	tinyint	Use volume once
UseCatalog	tinyint	Set to use catalog
AcceptAnyVolume	tinyint	Accept any volume from Pool
VolRetention	bigint	64 bit seconds to retain volume
AutoPrune	tinyint	Yes/no for autopruning
Recyle	tinyint	Yes/no for allowing auto recycling of Volume
PoolType	enum	Backup, Copy, Cloned, Archive, Migration
LabelFormat	Tinyblob	Label format

The **Pool** table contains one entry for each media pool controlled by Bacula in this database. One media record exists for each of the NumVols contained in the Pool. The PoolType is a Bacula defined keyword. The MediaType is defined by the administrator, and corresponds to the MediaType specified in the Director's Storage definition record. The CurrentVol is the sequence number of the Media record for the current volume.

Client		
Column Name	**Data Type**	**Remark**
ClientId	integer	Primary Key
Name	TinyBlob	File Services Name
UName	TinyBlob	uname −a from Client (not yet used)

AutoPrune	tinyint	Yes/no for autopruning
FileRetention	bigint	64 bit seconds to retain Files
JobRetention	bigint	64 bit seconds to retain Job

The **Client** table contains one entry for each machine backed up by Bacula in this database. Normally the Name is a fully qualified domain name.

Client		
Column Name	**Data Type**	**Remark**
VersionId	integer	Primary Key

The **Version** table defines the Bacula database version number. Bacula checks this number before reading the database to ensure that it is compatible with the Bacula binary file.

MySQL Table Definition

The commands used to create the MySQL tables are as follows:

```
USE bacula;
CREATE TABLE Filename (
  FilenameId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Name BLOB NOT NULL,
  PRIMARY KEY(FilenameId),
  INDEX (Name(30))
  );

CREATE TABLE Path (
   PathId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   Path BLOB NOT NULL,
   PRIMARY KEY(PathId),
   INDEX (Path(50))
   );


CREATE TABLE File (
   FileId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   FileIndex INTEGER UNSIGNED NOT NULL,
   JobId INTEGER UNSIGNED NOT NULL REFERENCES Job,
   PathId INTEGER UNSIGNED NOT NULL REFERENCES Path,
   FilenameId INTEGER NOT NULL REFERENCES Filename,
   MarkId INTEGER UNSIGNED NOT NULL DEFAULT 0,
   LStat TINYBLOB NOT NULL,
   MD5 TINYBLOB NOT NULL,
   PRIMARY KEY(FileId),
   INDEX (JobId),
   INDEX (PathId),
   INDEX (FilenameId)
   );
```

```
CREATE TABLE Job (
   JobId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   Job TINYBLOB NOT NULL,
   Name TINYBLOB NOT NULL,
   PurgedFiles TINYINT NOT NULL DEFAULT 0,
   Type BINARY(1) NOT NULL,
   Level BINARY(1) NOT NULL,
   ClientId INTEGER NOT NULL REFERENCES Client,
   JobStatus BINARY(1) NOT NULL,
   SchedTime DATETIME NOT NULL,
   StartTime DATETIME NOT NULL,
   EndTime DATETIME NOT NULL,
   JobTDate BIGINT UNSIGNED NOT NULL,
   VolSessionId INTEGER UNSIGNED NOT NULL,
   VolSessionTime INTEGER UNSIGNED NOT NULL,
   JobFiles INTEGER UNSIGNED NOT NULL,
   JobBytes BIGINT UNSIGNED NOT NULL,
   JobErrors INTEGER UNSIGNED NOT NULL,
   JobMissingFiles INTEGER UNSIGNED NOT NULL,
   PoolId INTEGER UNSIGNED NOT NULL REFERENCES Pool,
   FileSetId INTEGER UNSIGNED NOT NULL REFERENCES FileSet,
   PRIMARY KEY(JobId),
   INDEX (Name(128))
   );

CREATE TABLE FileSet (
   FileSetId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   FileSet TINYBLOB NOT NULL,
   MD5 TINYBLOB NOT NULL,
   PRIMARY KEY(FileSetId)
   );

CREATE TABLE JobMedia (
   JobMediaId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   JobId INTEGER UNSIGNED NOT NULL REFERENCES Job,
   MediaId INTEGER UNSIGNED NOT NULL REFERENCES Media,
   FirstIndex INTEGER UNSIGNED NOT NULL,
   LastIndex INTEGER UNSIGNED NOT NULL,
   StartFile INTEGER UNSIGNED NOT NULL,
   EndFile INTEGER UNSIGNED NOT NULL,
   StartBlock INTEGER UNSIGNED NOT NULL,
   EndBlock INTEGER UNSIGNED NOT NULL,
   PRIMARY KEY(JobMediaId),
   INDEX (JobId, MediaId)
   );


CREATE TABLE Media (
   MediaId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   VolumeName TINYBLOB NOT NULL,
   Slot INTEGER NOT NULL DEFAULT 0,
   PoolId INTEGER UNSIGNED NOT NULL REFERENCES Pool,
   MediaType TINYBLOB NOT NULL,
   FirstWritten DATETIME NOT NULL,
   LastWritten DATETIME NOT NULL,
   LabelDate DATETIME NOT NULL,
   VolJobs INTEGER UNSIGNED NOT NULL,
   VolFiles INTEGER UNSIGNED NOT NULL,
   VolBlocks INTEGER UNSIGNED NOT NULL,
```

```
   VolMounts INTEGER UNSIGNED NOT NULL,
   VolBytes BIGINT UNSIGNED NOT NULL,
   VolErrors INTEGER UNSIGNED NOT NULL,
   VolWrites INTEGER UNSIGNED NOT NULL,
   VolMaxBytes BIGINT UNSIGNED NOT NULL,
   VolCapacityBytes BIGINT UNSIGNED NOT NULL,
   VolStatus ENUM('Full', 'Archive', 'Append', 'Recycle', 'Purged',
    'Read-Only', 'Disabled', 'Error', 'Busy') NOT NULL,
   Recycle TINYINT NOT NULL,
   VolRetention BIGINT UNSIGNED NOT NULL,
   PRIMARY KEY(MediaId),
   INDEX (PoolId)
   );

CREATE TABLE Pool (
   PoolId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   Name TINYBLOB NOT NULL,
   NumVols INTEGER UNSIGNED NOT NULL,
   MaxVols INTEGER UNSIGNED NOT NULL,
   UseOnce TINYINT NOT NULL,
   UseCatalog TINYINT NOT NULL,
   AcceptAnyVolume TINYINT DEFAULT 0,
   VolRetention BIGINT UNSIGNED NOT NULL,
   AutoPrune TINYINT DEFAULT 0,
   Recycle TINYINT DEFAULT 0,
   PoolType ENUM('Backup', 'Copy', 'Cloned', 'Archive', 'Migration') NOT NULL,
   LabelFormat TINYBLOB,
   UNIQUE (Name(128)),
   PRIMARY KEY (PoolId)
   );


CREATE TABLE Client (
   ClientId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
   Name TINYBLOB NOT NULL,
   Uname TINYBLOB NOT NULL,       /* full uname -a of client */
   AutoPrune TINYINT DEFAULT 0,
   FileRetention BIGINT UNSIGNED NOT NULL,
   JobRetention  BIGINT UNSIGNED NOT NULL,
   UNIQUE (Name(128)),
   PRIMARY KEY(ClientId)
   );

CREATE TABLE Version (
   VersionId INTEGER UNSIGNED NOT NULL
   );

-- Initialize Version
INSERT INTO Version (VersionId) VALUES (2);

CREATE TABLE Counters (
   Counter TINYBLOB NOT NULL,
   PoolId INTEGER UNSIGNED NOT NULL REFERENCES Pool,
   MinValue INTEGER,
   MaxValue INTEGER,
   CurrentValue INTEGER,
   WrapCounter TINYBLOB NOT NULL
   );
```

Bacula Internal Component Designs

Bacula Internal Component Design Documents

- Developer Notes
- Intradaemon Protocols
- Storage Media Format
- Memory Management Design
- Bacula Network Protocol
- Our MD5 Algorithm
- Smart Memory Allocation Routines

Bacula Developer Notes

General

This document is intended mostly for developers and describes the the general framework of making Bacula source changes.

Contributions

Contributions from other programmers will be broken into two groups. The first are contributions that are aids and not essential to Bacula. In general, these will be scripts or will go into and examples or contributions directory.

The second class of contributions are those which will be integrated with Bacula and become an essential part. The rest of this document describes some of the requirements for such code.

Copyrights

To avoid future problems concerning changing licensing or copyrights, all code contributions must be in the Public Domain or have the copyright assigned to Kern Sibbald and John Walker as in the current code. The author of the code can be clearly designated. The purpose of this requirement is to avoid future problems as described above. To understand what I mean about future problems, please examine the difficulties Mozilla is having finding previous contributors at http://www.mozilla.org/MPL/missing.html.

Although the copyright will be held by Kern and John, each developer is expected to indicate that he wrote and/or modified a particular module (or file).

Although we have absolutely no plans for any commercial venture, it would be ashame to totally rule it out. Our desire is that if one day something commercial is developed around **Bacula** (as is the case with MySQL, RedHat, Ximian, and other organizations), each developer will have an opportunity to participate, at a minimum, proportional to his prior contributions. Again, we have no current plans for creating any such commercial enterprise.

If you have any doubts about this, please don't hesitate to ask. Our (John and mine) track records with Autodesk are easily available; early programmers/founders/contributors and later employees had substantial shares of the company, and no one founder had a controlling part of the company. Even though Microsoft created many millionaires among early employees, the politics of Autodesk (during our time at the helm) is in stark contrast to Microsoft where the majority of the company is still tightly held among a few.

Developing Bacula

Typically the simplest way to develop Bacula is to open one xterm window pointing to the source directory you wish to update; a second xterm window at the top source directory level, and a third xterm window at the bacula directory <top>/src/bacula. After making source changes in one of the directories, in the top source directory xterm, build the source, and start the daemons by entering:

make

and

./startit

then in the enter:

./console

or

./gnome−console

to start the Console program. Enter any commands for testing. For example: run kernsverify full.

Note, the instructions here to use **./startit** are different from using a production system where the administrator starts **Bacula** by entering **./bacula start**. This difference allows a development version of **Bacula** to be run on a computer at the same time that a production system is running. The **./startit** strip starts **Bacula** using a different set of configuration files, and thus permits avoiding conflicts with any production system.

To make additional source changes, exit from the Console program, and in the top source directory, stop the daemons by entering:

./stopit

then repeat the process.

Debugging

Probably the first thing to do is to turn on debug output.

A good place to start is with a debug level of 20 as in **./startit −d20**. The startit command starts all the daemons with the same debug level. Alternatively, you can start the appropriate daemon with the debug level you want. If you really need more info, a debug level of 60 is not bad, and for just about everything a level of 200.

Using a Debugger

If you have a serious problem such as a segmentation fault, it can usually be found quickly using a good multiple thread debugger such as **gdb**. For example, suppose you get a segmentation violation in **bacula−dir**. You might use the following to find the problem:

<start the Storage and File daemons>
cd dird
gdb ./bacula−dir
run −f −s −c ./dird.conf
<it dies with a segmentation fault>
where

The **−f** option is specified on the **run** command to inhibit **dird** from going into the background. You may also want to add the **−s** option to the run command to disable signals which can potentially interfere with the debugging.

As an alternative to using the debugger, each **Bacula** daemon has a built in back trace feature when a serious error is encountered. It calls the debugger on itself, produces a back trace, and emails the report to the developer. For more details on this, please see the chapter in this manual entitled What To Do When Bacula Crashes (Kaboom).

Memory Leaks

Because Bacula runs routinely and unattended on client and server machines, it may run for a long time. As a consequence, from the very beginning, Bacula uses SmartAlloc to ensure that there are no memory leaks. To make detection of memory leaks effective, all **Bacula** code that dynamically allocates memory MUST have a way to release it. In general when the memory is no longer needed, it should be immediately released, but in some cases, the memory will be held during the entire time that Bacula is executing. In that case, there MUST be a routine that can be called at termination time that releases the memory. In this way, we will be able to detect memory leaks. Be sure to immediately correct any and all memory leaks that are printed at the termination of the daemons.

Special Files

Kern uses files named 1, 2, ... 9 with any extension as scratch files. Thus any files with these names are subject to being rudely deleted at any time.

Indenting

Indenting is done three spaces at a time with braces on the first line. Please put braces in both the **if** and **else** clauses of all **if** statements. This avoids a lot of pain later with missed statements. This coding practice was not followed from the very beginning so when you find <naked> if statements, please correct them.

When Implementing Incomplete Code

Please identify all incomplete code with a comment that contains *****FIXME*****, where there are three asterisks (*) before and after the word FIXME (in capitals) and no intervening spaces. This is important as it allows new programmers to easily recognize where things are partially implemented.

Bacula Source File Structure

The distribution generally comes as a tar file of the form **bacula.x.y.z.tar.gz** where x, y, and z are the version, release, and update numbers respectively.

Once you detar this file, you will have a directory structure as follows:

```
|− bacula              (main source directory containing configuration
   |                   and installation files)
   |− autoconf         (automatic configuration files, not normally used
```

```
|                    by users)
|- doc              (documentation directory)
   | - home-page     (Bacula's home page source)
   | - html-manual   (html document directory)
   | - techlogs      (Technical development notes);
|- intl             (programs used to translate)
|- platforms        (OS specific installation files)
   | - redhat       (Red Hat instalation)
   | - solaris      (Sun installation)
   | - freebsd      (FreeBSD installation)
   | - irix         (Irix installation -- not tested)
   | - unknown      (Default if system not identified)
|- po               (translations of source strings)
|- src              (source directory; contains global header files)
   |- cats          (SQL catalog database interface directory)
   |- console       (bacula user agent directory)
   |- dird          (Director daemon)
   |- filed         (Unix File daemon)
      |- win32      (Win32 files to make bacula-fd be a service)

   |- findlib       (Unix file find library for File daemon)
   |- gnome-console (GNOME version of console program)
   |- immortal      (Cweb immortal code -- not currently used)
   |- lib           (General Bacula library)
   |- stored        (Storage daemon)
   |- tools         (Various tool programs)
```

Header Files

Please carefully follow the scheme defined below as it permits in general only two header file includes per C file, and thus vastly simplifies programming. With a large complex project like Bacula, it isn't always easy to ensure that the right headers are invoked in the right order (there are a few kludges to make this happen -- i.e. in a few include files because of the chicken and egg problem, certain references to typedefs had to be replaced with **void**).

Every file should include **bacula.h**. It pulls in just about everything, with very few exceptions. If you have system dependent ifdefing, please do it in **baconfig.h**. The version number and date are kept in **version.h**.

Each of the subdirectories (console, cats, dird, filed, findlib, lib, stored, ...) contains a single directory dependent include file generally the name of the directory, which should be included just after the include of **bacula.h**. This file (for example, for the dird directory, it is **dird.h**) contains either definitions of things generally needed in this directory, or it includes the appropriate header files. It always includes **protos.h**. See below.

Each subdirectory contains a header file named **protos.h**, which contains the prototypes for subroutines exported by files in that directory. **protos.h** is always included by the main directory dependent include file.

Programming Standards

For the most part, all code should be written in C unless there is a burning reason to use C++, and then only the simplest C++ constructs will be used.

Code should have some documentation −− not a lot, but enough so that I can understand it. Look at the current code, and you will see that I document more than most, but am definitely not a fanatic.

I like simple linear code where possible. Gotos are strongly discouraged except for handling an error to either bail out or to retry some code.

Indenting Standards

I cannot stand code indented 8 columns at a time. This makes it unreadable. Even 4 at a time uses a lot of space, so I have adopted indenting 3 spaces at every level. Braces are required in all if statements (missing in some very old code). To avoid generating too many lines, the first brace appears on the first line (e.g. of an if), and the closing brace is on a line by itself. E.g.

```
if (abc) {
   some_code;
}
```

Just follow the convention in the code. Originally I indented case clauses under a switch(), but now I prefer non−indented cases. Choose what you want. Avoid using // style comments except for temporary code. Standard C comments are preferred (this also keeps the code closer to C).

I'm flexible, but it prefer to generally keep the same style throughout the product. This will make it easier for everyone.

Message Classes

Currently, there are four classes of messages: Debug, Error, Job, and Memory.

Debug Messages

Debug messages are designed to be turned on at a specified debug level and are always sent to STDOUT. There are designed to only be used in the development debug process. They are coded as:

DmsgN(level, message, arg1, ...)

where the N is a number indicating how many arguments are to be substituted into the message (i.e. it is a count of the number arguments you have in your message −− generally the number of percent signs (%)). **level** is the debug level at which you wish the message to be printed. message is the debug message to be printed, and arg1, ... are the arguments to be substituted. Since not all compilers support #defines with varargs, you must explicitly specify how many arguments you have.

When the debug message is printed, it will automatically be prefixed by the name of the daemon which is running, the filename where the Dmsg is, and the line number within the file.

Some actual examples are:

Dmsg2(20, "MD5len=%d MD5=%s\n", strlen(buf), buf);

Dmsg1(9, "Created client %s record\n", client−>hdr.name);

Error Messages

Error messages are messages that are related to the daemon as a whole rather than a particular job. For example, an out of memory condition my generate an error message. They are coded as:

EmsgN(error−code, level, message, arg1, ...)

As with debug messages, you must explicitly code the of arguments to be substituted in the message. error−code indicates the severity or class of error, and it may be one of the following:

M_ABORT	Causes the daemon to immediately abort. This should be used only in extreme cases. It attempts to produce a traceback.
M_ERROR_TERM	Causes the daemon to immediately terminate. This should be used only in extreme cases. It does not produce a traceback.
M_FATAL	Causes the daemon to terminate the current job, but the daemon keeps running
M_ERROR	Reports the error. The daemon and the job continue running
M_WARNING	Reports an warning message. The daemon and the job continue running
M_INFO	Reports an informational message.

There are other error message classes, but they are in a state of being redesigned or deprecated, so please do not use them. Some actual examples are:

Emsg1(M_ABORT, 0, "Cannot create message thread: %s\n", strerror(status));

Emsg3(M_WARNING, 0, "Connect to File daemon %s at %s:%d failed. Retrying ...\n", client−>hdr.name, client−>address, client−>port);

Emsg3(M_FATAL, 0, "bdird<filed: bad response from Filed to %s command: %d %s\n", cmd, n, strerror(errno));

Job Messages

Job messages are messages that pertain to a particular job such as a file that could not be saved, or the number of files and bytes that were saved.

Memory Messages

Memory messages are messages that are edited into a memory buffer. Generally they are used in low level routines such as the low level device file dev.c in the Storage daemon or in the low level Catalog routines. These routines do not generally have access to the Job Control Record and so they return error messages reformatted in a memory buffer. Mmsg() is the way to do this.

Bacula Porting Notes

General

This document is intended mostly for developers who wish to port **Bacula** to a system that is not **officially** supported.

It is hoped that **Bacula** clients will eventually run on every imaginable system that needs backing up (perhaps even a Palm). It is also hoped that the **Bacula** Directory and Storage daemons will run on every system capable of supporting them.

Requirements

In General, the following holds true:

- **Bacula** has been compiled and run on Linux RedHat, FreeBSD, and Solaris systems.
- In addition, clients exist on Win32 (Cygwin), and Irix
- It requires GNU C++ to compile. You can try with other compilers, but you are on your own. The Irix client is built with the Irix complier, but, in general, you will need GNU.
- Your compiler must provide support for 64 bit signed and unsigned integers.
- You will need a recent copy of the **autoconf** tools loaded on your system (version 2.13 or later). The **autoconf** tools are used to build the configuration program, but are not part of the **Bacula** source distribution.
- There are certain third party packages that **Bacula** needs. Except for MySQL, they can all be found in the **depkgs** and **depkgs1** releases.
- If you want to build the Win32 binaries, you will need the full Cygwin 1.3.10 release. Although all components build (console has some warnings), only the File daemon has been tested. Please note that if you attempt to build Bacula on any other version of Cygwin, particularly previous versions, you will be on your own.
- **Bacula** requires a good implementation of pthreads to work.
- The source code has been written with portability in mind and is mostly POSIX compatible. Thus porting to any POSIX compatible operating system should be relatively easy.

Steps to Take

- The first step is to ensure that you have version 2.13 or later of the **autoconf** tools loaded. You can skip this step, but making changes to the configuration program will be difficult or impossible.
- The run a **./configure** command in the main source directory and examine the output. It should look something like the following:

```
Configuration on Mon Oct 28 11:42:27 CET 2002:

  Host:                                   i686-pc-linux-gnu -- redhat 7.3
  Bacula version:                         1.27 (26 October 2002)
  Source code location:                   .
  Install binaries:                       /sbin
  Install config files:                   /etc/bacula
  C Compiler:                             gcc
```

```
C++ Compiler:                         c++
Compiler flags:                       -g -O2
Linker flags:
Libraries:                            -lpthread
Statically Linked Tools:              no
Database found:                       no
Database type:                        Internal
Database lib:

Job Output Email:                     root@localhost
Traceback Email:                      root@localhost
SMTP Host Address:                    localhost
Director Port                         9101
File daemon Port                      9102
Storage daemon Port                   9103
Working directory                     /etc/bacula/working
SQL binaries Directory

Large file support:                   yes
readline support:                     yes
cweb support:                         yes /home/kern/bacula/depkgs/cweb
TCP Wrappers support:                 no
ZLIB support:                         yes
enable-smartalloc:                    yes
enable-gnome:                         no
gmp support:                          yes
```

The details depend on your system. The first thing to check is that it properly identified your host on the **Host:** line. The first part (added in version 1.27) is the GNU four part identification of your system. The part after the -- is your system and the system version. Generally, if your system is not yet supported, you must correct these.

- If the **./configure** does not function properly, you must determine the cause and fix it. Generally, it will be because some required system routine is not available on your machine.
- To correct problems with detection of your system type or with routines and libraries, you must edit the file **<bacula-src>/autoconf/configure.in**. This is the "source" from which **configure** is built. In general, most of the changes for your system will be made in **autoconf/aclocal.m4** in the routine **BA_CHECK_OPSYS** or in the routine **BA_CHECK_OPSYS_DISTNAME**. I have already added the necessary code for most systems, but if yours shows up as **unknown** you will need to make changes. Then as mentioned above, you will need to set a number of system dependent items in **configure.in** in the **case** statement at approximately line 1050 (depending on the **Bacula** release.
- The items to in the case statement that corresponds to your system are the following:
    - DISTVER -- set to the version of your operating system. Typically some form of **uname** obtains it.
    - TAPEDRIVE -- the default tape drive. Not too important as the user can set it as an option.
    - PSCMD -- set to the **ps** command that will provide the PID in the first field and the program name in the second field. If this is not set properly, the **bacula stop** script will most likely not be able to stop **Bacula** in all cases.
    - hostname -- command to return the base host name (non-qualified) of your system. This is generally the machine name. Not too important as the user can correct this in his configuration file.

- CFLAGS −− set any special compiler flags needed. Many systems need a special flag to make pthreads work. See cygwin for an example.
- LDFLAGS −− set any special loader flags. See cygwin for an example.
- PTHREAD_LIB −− set for any special pthreads flags needed during linking. See freebsd as an example.
- lld −− set so that a "long long int" will be properly edited in a printf() call.
- llu −− set so that a "long long unsigned" will be properly edited in a printf() call.
- PFILES −− set to add any files that you may define is your platform subdirectory. These files are used for installation of automatic system startup of **Bacula** daemons.

- To rebuild a new version of **configure** from a changed **autoconf/configure.in** you enter **make configure** in the top level **Bacula** source directory. You must have done a ./configure prior to trying to rebuild the configure script or it will get into an infinite loop.
- After you have a working **configure** script, you may need to make a few system dependent changes to the way **Bacula** works. Generally, these are done in **src/baconfig.h**. You can find a few examples of system dependent changes towards the end of this file. For example, on Irix systems, there is no definition for **socklen_t**, so it is made in this file. If your system has structure alignment requirements, check the definition of BALIGN in this file. Currently, all Bacula memory is aligned on a **double** boundry.
- If you are having problems with Bacula's type definitions, you might look at **src/bc_types.h** where all the types such as **uint32_t**, **uint64_t**, etc. that Bacula uses are defined.


Bacula Developer Notes

Index

Bacula Daemon Protocol


Porting Notes

Index

Storage Media Output Format

Daemon Protocol

General

This document describes the protocols used between the various daemons. As Bacula has developed, it has become quite out of date. The general idea still holds true, but the details of the fields for each command, and indeed the commands themselves have changed considerably.

It is intended to be a technical discussion of the general daemon protocols and as such is not targeted at end users but rather at developers and system administrators that want or need to know more of the working details of **Bacula**.

Low Level Network Protocol

At the lowest level, the network protocol is handled by **BSOCK** packets which contain a lot of information about the status of the network connection: who is at the other end, etc. Each basic **Bacula** network read or write actually consists of two low level network read/writes. The first write always sends four bytes of data in machine independent byte order. If data is to follow, the first four bytes are a positive non−zero integer indicating the length of the data that follow in the subsequent write. If the four byte integer is zero or negative, it indicates a special request, a sort of network signaling capability. In this case, no data packet will follow. The low level BSOCK routines expect that only a single thread is accessing the socket at a time. It is advised that multiple threads do not read/write the same socket. If you must do this, you must provide some sort of locking mechanism. I would not be appropriate for efficiency reasons to make every call to the BSOCK routines lock and unlock the packet.

General Daemon Protocol

In general, all the daemons follow the following global rules. There may be exceptions depending on the specific case. Normally, one daemon will be sending commands to another daemon (specifically, the Director to the Storage daemon and the Director to the File daemon).

- Commands are always ASCII commands that are upper/lower case dependent as well as space sensitive.
- All binary data is converted into ASCII (either with printf statements or using base64 encoding).
- All responses to commands sent are always prefixed with a return numeric code where codes in the 1000's are reserved for the Director, the 2000's are reserved for the File daemon, and the 3000's are reserved for the Storage daemon.
- Any response that is not prefixed with a numeric code is a command (or subcommand if you like) coming from the other end. For example, while the Director is corresponding with the Storage daemon, the Storage daemon can request Catalog services from the Director. This convention permits each side to send commands to the other daemon while simultaneously responding to commands.
- Any response that is of zero length, depending on the context, either terminates the data stream being sent or terminates command mode prior to closing the connection.
- Any response that is of negative length is a special sign that normally requires a response. For example, during data transfer from the File daemon to the Storage daemon, normally the File daemon sends continuously without intervening reads. However,

periodically, the File daemon will send a packet of length −1 indicating that the current data stream is complete and that the Storage daemon should respond to the packet with an OK, ABORT JOB, PAUSE, etc. This permits the File daemon to efficiently send data while at the same time occasionally "polling" the Storage daemon for his status or any special requests.

Currently, these negative lengths are specific to the daemon, but shortly, the range 0 to −999 will be standard daemon wide signals, while −1000 to −1999 will be for Director user, −2000 to −2999 for the File daemon, and −3000 to −3999 for the Storage daemon.

The Protocol Used Between the Director and the Storage Daemon

Before sending commands to the File daemon, the Director opens a Message channel with the Storage daemon, identifies itself and presents its password. If the password check is OK, the Storage daemon accepts the Director. The Director then passes the Storage daemon, the JobId to be run as well as the File daemon authorization (append, read all, or read for a specific session). The Storage daemon will then pass back to the Director a enabling key for this JobId that must be presented by the File daemon when opening the job. Until this process is complete, the Storage daemon is not available for use by File daemons.

```
SD: listens
DR: makes connection
DR: Hello <Director−name> calling <password>
SD: 3000 OK Hello
DR: JobId=nnn Allow=(append, read) Session=(*, SessionId)
                    (Session not implemented yet)
SD: 3000 OK Job Authorization=<password>
DR: use device=<device−name> media_type=<media−type>        pool_name=<pool−name> pool_typ
SD: 3000 OK use device
```

For the Director to be authorized, the <Director−name> and the <password> must match the values in one of the Storage daemon's Director resources (there may be several Directors that can access a single Storage daemon).

The Protocol Used Between the Director and the File Daemon

A typical conversation might look like the following:

```
FD: listens
DR: makes connection
DR: Hello <Director−name> calling <password>
FD: 2000 OK Hello
DR: JobId=nnn Authorization=<password>
FD: 2000 OK Job
DR: storage address = <Storage daemon address> port = <port−number>
          name = <DeviceName> mediatype = <MediaType>
FD: 2000 OK storage
DR: include
DR: <directory1>
DR: <directory2>
    ...
```

```
DR: Null packet
FD: 2000 OK include
DR: exclude
DR: <directory1>
DR: <directory2>
    ...
DR: Null packet
FD: 2000 OK exclude
DR: full
FD: 2000 OK full
DR: save
FD: 2000 OK save
FD: Attribute record for each file as sent to the
    Storage daemon (described above).
FD: Null packet
FD: <append close responses from Storage daemon>
    e.g.
    3000 OK Volumes = <number of volumes>
    3001 Volume = <volume-id> <start file> <start block>
         <end file> <end block> <volume session-id>
    3002 Volume data = <date/time of last write> <Number bytes written>
         <number errors>
    ... additional Volume / Volume data pairs for volumes 2 .. n
FD: Null packet

FD: close socket
```

The Save Protocol Between the File Daemon and the Storage Daemon

Once the Storage daemon has issued the **save** command, the File daemon will contact the Storage daemon to begin the save.

In what follows: FD: refers to information set via the network from the File daemon to the Storage daemon, and SD: refers to information set from the Storage daemon to the File daemon.

Command and Control Information

Command and control information is exchanged in human readable ASCII commands.

```
FD: listens
SD: makes connection
FD: append open session = <JobId> [<password>]
SD: 3000 OK ticket = <number>
FD: append data <ticket-number>
SD: 3000 OK data address = <IPaddress> port = <port>
```

Data Information

The Data information consists of the file attributes and data to the Storage daemon. For the most part, the data information is sent one way: from the File daemon to the Storage daemon. This allows the File daemon to transfer information as fast as possible without a lot of handshaking and network overhead.

However, from time to time, the File daemon needs to do a sort of checkpoint of the situation to ensure that everything is going well with the Storage daemon. To do so, the File daemon sends a packet with a negative length indicating that he wishes the Storage daemon to respond by sending a packet of information to the File daemon. The File daemon then waits to receive a packet from the Storage daemon before continuing.

All data sent are in binary format except for the header packet, which is in ASCII. There are two packet types used data transfer mode: a header packet, the contents of which are known to the Storage daemon, and a data packet, the contents of which are never examined by the Storage daemon.

The first data packet to the Storage daemon will be an ASCII header packet consisting of the following data.

<File−Index> <Stream−Id> <Info>

where **<File−Index>** is a sequential number beginning from one that increments with each file (or directory) sent.

where **<Stream−Id>** will be 1 for the Attributes record and 2 for uncompressed File data. 3 is reserved for the MD5 signature for the file.

where **<Info>** transmit information about the Stream to the Storage Daemon. It is a character string field where each character has a meaning. The only character currently defined is 0 (zero), which is simply a place holder (a no op). In the future, there will be codes indicating compressed data, encrypted data, etc.

Immediately following the header packet, the Storage daemon will expect any number of data packets. The series of data packets is terminated by a zero length packet, which indicates to the Storage daemon that the next packet will be another header packet. As previously mentioned, a negative length packet is a request for the Storage daemon to temporarily enter command mode and send a reply to the File daemon. Thus an actual conversation might contain the following exchanges:

```
FD: <1 1 0> (header packet)
FD: <data packet containing file−attributes>
FD: Null packet
FD: <1 2 0>
FD: <multiple data packets containing the file data>
FD: Packet length = −1
SD: 3000 OK
FD: <2 1 0>
FD: <data packet containing file−attributes>
FD: Null packet
FD: <2 2 0>
FD: <multiple data packets containing the file data>
FD: Null packet
FD: Null packet


FD: append end session <ticket−number>
SD: 3000 OK end
FD: append close session <ticket−number>
SD: 3000 OK Volumes = <number of volumes>
```

```
SD: 3001 Volume = <volumeid> <start file> <start block>
     <end file> <end block> <volume session-id>
SD: 3002 Volume data = <date/time of last write> <Number bytes written>
     <number errors>
SD: ... additional Volume / Volume data pairs for
     volumes 2 .. n
FD: close socket
```

The information returned to the File daemon by the Storage daemon in response to the **append close session** is transmit in turn to the Director.

  
Porting Notes Index Storage Media Output Format

Storage Media Output Format

General

This document describes the media format written by the Storage daemon. The Storage daemon reads and writes in units of blocks. Blocks contain records. Each block has a block header followed by records, and each record has a record header followed by record data.

This chapter is intended to be a technical discussion of the Media Format and as such is not targeted at end users but rather at developers and system administrators that want or need to know more of the working details of **Bacula**.

Definitions

Block
: A block represents the primitive unit of information that the Storage daemon reads and writes to a physical device. Normally, for a tape device, it will be the same as a tape block. The Storage daemon always reads and writes blocks. A block consists of block header information followed by records. Clients of the Storage daemon (the File daemon) normally never see blocks. However, some of the Storage tools (bls, bscan, bextract, ...) may be use block header information.

Record
: A record consists of a Record Header, which is managed by the Storage daemon and Record Data, which is the data received from the Client. The details are described below.

JobId
: A number assigned by the Director daemon for a particular job. This number will be unique for that particular Director (Catalog). The daemons use this number to keep track of individual jobs. Within the Storage daemon, the JobId may not be unique if several Directors are accessing the Storage daemon simultaneously.

Session
: A Session corresponds to a Job with the exception that each session is uniquely identified within the Storage daemon (see below).

VolSessionId
: A unique number assigned by the Storage daemon to a particular session it is having with a File daemon. This number by itself is not unique to the given Volume, but with the VolSessionTime, it is unique.

VolSessionTime
: A unique number assigned by the Storage daemon to a particular Storage daemon execution. It is actually the Unix time_t value of when the Storage daemon began execution cast to a 32 bit unsigned integer. The combination of the **VolSessionId** and the **VolSessionTime** for a given Storage daemon is guaranteed to be unique for each Job (or session).

FileIndex
: A sequential number beginning at one assigned by the File daemon to the files within a job that are sent to the Storage daemon for backup. The Storage daemon ensures that this number is greater than zero and sequential. Note, the Storage daemon uses negative FileIndexes to flag Session Start and End Labels as well as End of Volume Labels.

Stream

An arbitrary number assigned by the File daemon to the parts (attributes, data, ...) of a file that are sent to the Storage daemon. The Storage daemon has no knowledge of the details of a Stream; it simply represents a numbered stream of bytes. The data for a given stream may be passed to the Storage daemon in multiple packets.

Block Header

A block header consists of a block identification ("BB01"), a block length in bytes (typically 64,512) a checksum, and sequential block number. Each block starts with a Block Header and is followed by Records.

Record Header

A record header contains the Volume Session Id, the Volume Session Time, the FileIndex, the Stream, and the size of the data record which follows. The Record Header is always immediately followed by a Data Record if the size given in the Header is greater than zero. Note, for Block headers of level BB02 (version 1.27 and later), the Record header does not contain the Volume Session Id and the Volume Session Time as these two fields are stored in the BB02 Block header.

Data Record

A data record consists of a binary stream of bytes and is always preceded by a Record Header.

Volume Label

A label placed by the Storage daemon at the beginning of each storage volume. It contains general information about the volume. It is written in Record format.

Begin Session Label

The Begin Session Label is a special record placed by the Storage daemon on the storage medium as the first record of an append session job with a File daemon. This record is useful for finding the beginning of a particular session (Job), since no records with the same VolSessionId and VolSessionTime will precede this record. This record is not normally visible outside of the Storage daemon. The Begin Session Label is similar to the Volume Label except that it contains additional information pertaining to the Session.

End Session Label

The End Session Label is a special record placed by the Storage daemon on the storage medium as the last record of an append session job with a File daemon. The End Session Record is distinguished by a FileIndex with a value of minus two (−2). This record is useful for detecting the end of a particular session since no records with the same VolSessionId and VolSessionTime will follow this record. This record is not normally visible outside of the Storage daemon. The End Session Label is similar to the Volume Label except that it contains additional information pertaining to the Session.

**End of Volume Label*

NOT YET IMPLEMENTED! The End of Volume label is a special record placed by the Storage daemon on the storage medium as the last record on the medium. It is somewhat analogous to the Volume Label except that it appears at the end of the volume. The End of Volume Label is distinguished by a File−Index with a value of minus three (−3). This record is not visible outside of the Storage daemon. It should be noted that this record is not guaranteed to be on every volume because on certain media such as old streaming tapes, there is no logical end of tape. As a consequence, when Bacula is writing the tape and detects the end of tape condition, on these older tapes, Bacula will be unable to write additional data to the tape.

Storage Daemon File Output Format

The file storage and tape storage formats are identical except that tape records are by default blocked into blocks of 64,512 bytes, except for the last block, which is the actual number of bytes written rounded up to a multiple of 1024 whereas the last record of file storage is not rounded up. The default block size of 64,512 bytes may be overridden by the user (some older tape drives only support block sizes of 32K). Each Session written to tape is terminated with an End of File mark (this will be removed later). Sessions written to file are simply appended to the end of the file.

Overall Format

A Bacula output file consists of Blocks of data. Each block contains a block header followed by records. Each record consists of a record header followed by the record data. The first record on a tape will always be the Volume Label Record. Wherever possible, the last record on the tape will be an End of Volume Label Record (not yet implemented −− version 1.26) . However this is not guaranteed as some tapes do not permit writing data after the End of Tape condition is signaled. Bacula will never attempt to read past an End of Volume Label.

No Record Header will be split across Bacula blocks. However, Record Data may be split across any number of Bacula blocks. Obviously this will not be the case for the Volume Label which will always be smaller than the Bacula Block size.

To simplify reading tapes, the Start of Session (SOS) and End of Session (EOS) records are never split across blocks. If this is about to happen, Bacula will write a short block before writing the session record (actually, the SOS record should always be the first record in a block, excepting perhaps the Volume label).

Due to hardware limitations, the last block written to the tape may not be fully written. *NOT YET IMPLEMENTED! In this case, Bacula will attempt to write an additional block containing only the End of Volume Label followed by an End of File mark. Due to hardware limitations, the End of Volume Label and the End of File mark may not be written.

When a new tape is mounted Bacula will write the full contents of the partially written block to the new tape ensuring that there is no loss of data. When reading a tape, Bacula will discard any block that is not totally written, thus ensuring that there is no duplication of data. In addition, since Bacula blocks are sequentially numbered within a Job, it is easy to ensure that no block is missing.

Serialization

All Block Headers, Record Headers, and Label Records are written using Bacula's serialization routines. These routines guarantee that the data is written to the output volume in a machine independent format.

Block Header

The format of the Block Header (version 1.27 and later) is:

```
uint32_t CheckSum;                /* Block check sum */
uint32_t BlockSize;               /* Block byte size including the header */
uint32_t BlockNumber;             /* Block number */
char ID[4] = "BB02";              /* Identification and block level */
uint32_t VolSessionId;            /* Session Id for Job */
uint32_t VolSessionTime;          /* Session Time for Job */
```

The format of the Block Header (version 1.26 and earlier) is:

```
uint32_t CheckSum;                /* Block check sum */
uint32_t BlockSize;               /* Block byte size including the header */
uint32_t BlockNumber;             /* Block number */
char ID[4] = "BB01";              /* Identification and block level */
```

The Block header is a fixed length and fixed format and is followed by Record Headers and Record Data. The CheckSum field is a 32 bit checksum of the block data and the block header but not including the CheckSum field. The Block Header is always immediately followed by a Record Header. If the tape is damaged, a Bacula utility will be able to recover as much information as possible from the tape by recovering blocks which are valid. The Block header is written using the Bacula serialization routines and thus is guaranteed to be in machine independent format. See below for version 2 of the block header.

Record Header

Each binary data record is preceded by a Record Header. The Record Header is fixed length and fixed format, whereas the binary data record is of variable length. The Record Header is written using the Bacula serialization routines and thus is guaranteed to be in machine independent format.

The format of the Record Header (version 1.27 or later) is:

```
int32_t FileIndex;                /* File index supplied by File daemon */
int32_t Stream;                   /* Stream number supplied by File daemon */
uint32_t DataSize;                /* size of following data record in bytes */
```

The format of the Record Header (version 1.26 or earlier) is:

```
uint32_t VolSessionId;            /* Unique ID for this session */
uint32_t VolSessionTime;          /* Start time/date of session */
int32_t FileIndex;                /* File index supplied by File daemon */
int32_t Stream;                   /* Stream number supplied by File daemon */
uint32_t DataSize;                /* size of following data record in bytes */
```

This record is followed by the binary Stream data of DataSize bytes, followed by another Record Header record and the binary stream data. For the definitive definition of this record, see record.h in the src/stored directory.

Additional notes on the above:

*The **VolSessionId***
: is a unique sequential number that is assigned by the Storage Daemon to a particular Job. This number is sequential since the start of execution of the daemon.

*The **VolSessionTime***

is the time/date that the current execution of the Storage Daemon started. It assures that the combination of VolSessionId and VolSessionTime is unique for every jobs written to the tape, even if there was a machine crash between two writes.

*The **FileIndex***
is a sequential file number within a job. The Storage daemon requires this index to be greater than zero and sequential. Note, however, that the File daemon may send multiple Streams for the same FileIndex. In addition, the Storage daemon uses negative FileIndices to hold the Begin Session Label, the End Session Label, and the End of Volume Label.

*The **Stream***
is defined by the File daemon and is intended to be used to identify separate parts of the data saved for each file (attributes, file data, ...). The Storage Daemon has no idea of what a Stream is or what it contains except that the Stream is required to be a positive integer. Negative Stream numbers are used internally by the Storage daemon to indicate that the record is a continuation of the previous record (the previous record would not entirely fit in the block).
For Start Session and End Session Labels (where the FileIndex is negative), the Storage daemon uses the Stream field to contain the JobId.

*The **DataSize***
is the size in bytes of the binary data record that follows the Session Record header. The Storage Daemon has no idea of the actual contents of the binary data record. For standard Unix files, the data record typically contains the file attributes or the file data. For a sparse file (not yet implemented), the binary data record may have a sub−header which indicates the storage address (probably 64 bits) for the data block.

The Record Header is never split across two blocks. If there is not enough room in a block for the full Record Header, the block is padded to the end with zeros and the Record Header begins in the next block. The data record, on the other hand, may be split across multiple blocks and even multiple physical volumes. When a data record is split, the second (and possibly subsequent) piece of the data is preceded by a new Record Header. Thus each piece of data is always immediately preceded by a Record Header. When reading a record, if Bacula finds only part of the data in the first record, it will automatically read the next record and concatenate the data record to form a full data record.

Version BB02 Block Header

The existing block header BB01 was designed to hold records from multiple sessions. However, it is simpler (and probably more efficient) for each session (Job) to have its own private block. As a consequence, the SessionId and SessionTime can be written once in each Block Header and not in the Record Header. So, the second version of the Block Header is:

```
uint32_t CheckSum;              /* Block check sum */
uint32_t BlockSize;             /* Block byte size including the header */
uint32_t BlockNumber;           /* Block number */
char ID[4] = "BB02";            /* Identification and block level */
uint32_t VolSessionId;          /* Applies to all records  */
uint32_t VolSessionTime;        /*   contained in this block */
```

As with the previous version, the BB02 Block header is a fixed length and fixed format and is followed by Record Headers and Record Data. The CheckSum field is a 32 bit checksum of the

block data and the block header but not including the CheckSum field. The Block Header is always immediately followed by a Record Header. If the tape is damaged, a Bacula utility will be able to recover as much information as possible from the tape by recovering blocks which are valid. The Block header is written using the Bacula serialization routines and thus is guaranteed to be in machine independent format.

Version 2 Record Header

Version 2 Record Header is written to the medium when using Version BB02 Block Headers. The memory representation of the record is identical to the BB01 Record Header, but on the storage medium, the first two fields, namely VolSessionId and VolSessionTime will not be written. The Block Header will be filled with these values when the First user record is written (i.e. non label record) so that when the block is written, it will have the current and unique VolSessionId and VolSessionTime. On reading each record from the Block, the VolSessionId and VolSessionTime will be filled in the Record Header from the Block Header.

Volume Label Format

Tape volume labels are created by the Storage daemon in response to a **label** command given to the Console program, or alternatively by the **btape** program. created. Each volume is labeled with the following information using the Bacula serialization routines, which guarantee machine byte order independence.

For Bacula versions 1.27 and later, the Volume Label Format is:

```
char Id[32];                        /* Bacula 1.0 Immortal\n */
uint32_t VerNum;                    /* Label version number */

/* VerNum 11 and greater Bacula 1.27 and later */
btime_t   label_btime;              /* Time/date tape labeled */
btime_t   write_btime;              /* Time/date tape first written */

/* The following are 0 in VerNum 11 and greater */
float64_t write_date;               /* Date this label written */
float64_t write_time;               /* Time this label written */

char VolName[128];                  /* Volume name */
char PrevVolName[128];              /* Previous Volume Name */
char PoolName[128];                 /* Pool name */
char PoolType[128];                 /* Pool type */
char MediaType[128];                /* Type of this media */

char HostName[128];                 /* Host name of writing computer */
char LabelProg[32];                 /* Label program name */
char ProgVersion[32];               /* Program version */
char ProgDate[32];                  /* Program build date/time */
```

For Bacula versions 1.26 and earlier, the Volume Label is:

```
char Id[32];                        /* Bacula 0.9 mortal\n */
uint32_t VerNum;                    /* Label version number */

float64_t label_date;               /* Date tape labeled */
float64_t label_time;               /* Time tape labeled */
```

```
float64_t write_date;              /* Date this label written */
float64_t write_time;              /* Time this label written */

char VolName[128];                 /* Volume name */
char PrevVolName[128];             /* Previous Volume Name */
char PoolName[128];                /* Pool name */
char PoolType[128];                /* Pool type */
char MediaType[128];               /* Type of this media */

char HostName[128];                /* Host name of writing computer */
char LabelProg[32];                /* Label program name */
char ProgVersion[32];              /* Program version */
char ProgDate[32];                 /* Program build date/time */
```

Note, the LabelType (Volume Label, Volume PreLabel, Session Start Label, ...) is stored in the record FileIndex field of the Record Header and does not appear in the data part of the record.

Session Label

The Session Label is written at the beginning and end of each session as well as the last record on the physical medium. It has the following binary format:

```
char Id[32];                       /* Bacula Immortal ... */
uint32_t VerNum;                   /* Label version number */

uint32_t JobId;                    /* Job id */
uint32_t VolumeIndex;              /* sequence no of vol */

/* Prior to VerNum 11 */
float64_t write_date;              /* Date this label written */

/* VerNum 11 and greater */
btime_t   write_btime;             /* time/date record written */

/* The following is zero VerNum 11 and greater */
float64_t write_time;              /* Time this label written */

char PoolName[128];                /* Pool name */
char PoolType[128];                /* Pool type */
char JobName[128];                 /* base Job name */
char ClientName[128];
/* Added in VerNum 10 */
char Job[128];                     /* Unique Job name */
char FileSetName[128];             /* FileSet name */
uint32_t JobType;
uint32_t JobLevel;
```

In addition, the EOS label contains:

```
/* The remainder are part of EOS label only */
uint32_t JobFiles;
uint64_t JobBytes;
uint32_t start_block;
uint32_t end_block;
uint32_t start_file;
uint32_t end_file;
uint32_t JobErrors;
```

In addition, for VerNum greater than 10, the EOS label contains (in addition to the above):

```
uint32_t JobStatus                  /* Job termination code */
```

: Note, the LabelType (Volume Label, Volume PreLabel, Session Start Label, ...) is stored in the record FileIndex field and does not appear in the data part of the record. Also, the Stream field of the Record Header contains the JobId. This permits quick filtering without actually reading all the session data in many cases.

Overall Storage Format

```
            Current Bacula Tape Format
                  6 June 2001

         Version BB02 added 28 September 2002


   A Bacula tape is composed of tape Blocks.  Each block
     has a Block header followed by the block data. Block
     Data consists of Records. Records consist of Record
     Headers followed by Record Data.

   :=================================================:
   |                                                 |
   |                 Block Header (16 bytes)         |
   |                 (24 bytes version BB02)         |
   |-------------------------------------------------|
   |                                                 |
   |              Record Header (20 bytes)           |
   |               (12 bytes version BB02)           |
   |-------------------------------------------------|
   |                                                 |
   |                  Record Data                    |
   |                                                 |
   |-------------------------------------------------|
   |                                                 |
   |              Record Header (20 bytes)           |
   |               (12 bytes version BB02)           |
   |-------------------------------------------------|
   |                                                 |
   |                      ...                        |


   Block Header: the first item in each block. The format is
     shown below.

   Partial Data block: occurs if the data from a previous
     block spills over to this block (the normal case except
     for the first block on a tape). However, this partial
         data block is always preceded by a record header.

   Record Header: identifies the Volume Session, the Stream
     and the following Record Data size. See below for format.
```

Record data: arbitrary binary data.

Block Header Format BB01

CheckSum	(uint32_t)	
BlockSize	(uint32_t)	
BlockNumber	(uint32_t)	
"BB01"	(char [4])	

BBO1: Serves to identify the block as a
Bacula block and also servers as a block format identifier
should we ever need to change the format.

Block Header Format BB02

CheckSum	(uint32_t)	
BlockSize	(uint32_t)	
BlockNumber	(uint32_t)	
"BB02"	(char [4])	
VolSessionId	(uint32_t)	
VolSessionTime	(uint32_t)	

BBO2: Serves to identify the block as a
Bacula block and also servers as a block format identifier
should we ever need to change the format.

BlockSize: is the size in bytes of the block. When reading
back a block, if the BlockSize does not agree with the
actual size read, Bacula discards the block.

CheckSum: a checksum for the Block.

BlockNumber: is the sequential block number on the tape.

VolSessionId: a unique sequential number that is assigned
by the Storage Daemon to a particular Job.
This number is sequential since the start
of execution of the daemon.

VolSessionTime: the time/date that the current execution
of the Storage Daemon started. It assures
that the combination of VolSessionId and
VolSessionTime is unique for all jobs
written to the tape, even if there was a
machine crash between two writes.

Record Header Format BB01

VolSessionId	(uint32_t)	
VolSessionTime	(uint32_t)	

```
|-----------------------------------------------------|
|              FileIndex        (int32_t)             |
|-----------------------------------------------------|
|              Stream           (int32_t)             |
|-----------------------------------------------------|
|              DataSize         (uint32_t)            |
:=====================================================:

              Record Header Format BB02
:=====================================================:
|              FileIndex        (int32_t)             |
|-----------------------------------------------------|
|              Stream           (int32_t)             |
|-----------------------------------------------------|
|              DataSize         (uint32_t)            |
:=====================================================:
```

VolSessionId: a unique sequential number that is assigned by the Storage Daemon to a particular Job. This number is sequential since the start of execution of the daemon.

VolSessionTime: the time/date that the current execution of the Storage Daemon started. It assures that the combination of VolSessionId and VolSessionTime is unique for all jobs written to the tape, even if there was a machine crash between two writes.

FileIndex: a sequential file number within a job. The Storage daemon enforces this index to be greater than zero and sequential. Note, however, that the File daemon may send multiple Streams for the same FileIndex. The Storage Daemon uses negative FileIndices to identify Session Start and End labels as well as the End of Volume labels.

Stream: defined by the File daemon and is intended to be used to identify separate parts of the data saved for each file (attributes, file data, ...). The Storage Daemon has no idea of what a Stream is or what it contains.

DataSize: the size in bytes of the binary data record that follows the Session Record header. The Storage Daemon has no idea of the actual contents of the binary data record. For standard Unix files, the data record typically contains the file attributes or the file data. For a sparse file (not yet implemented), the binary data record my have a sub−header which indicates the storage address (probably 64 bits) for the data block.

```
                    Volume Label
:=====================================================:
|              Id               (32 bytes)            |
|-----------------------------------------------------|
```

```
|              VerNum           (uint32_t)              |
|-------------------------------------------------------|
|              label_date       (float64_t)             |
|              label_btime      (btime_t VerNum 11      |
|-------------------------------------------------------|
|              label_time       (float64_t)             |
|              write_btime      (btime_t VerNum 11      |
|-------------------------------------------------------|
|              write_date       (float64_t)             |
|                  0            (float64_t) VerNum 11   |
|-------------------------------------------------------|
|              write_time       (float64_t)             |
|                  0            (float64_t) VerNum 11   |
|-------------------------------------------------------|
|              VolName          (128 bytes)             |
|-------------------------------------------------------|
|              PrevVolName      (128 bytes)             |
|-------------------------------------------------------|
|              PoolName         (128 bytes)             |
|-------------------------------------------------------|
|              PoolType         (128 bytes)             |
|-------------------------------------------------------|
|              MediaType        (128 bytes)             |
|-------------------------------------------------------|
|              HostName         (128 bytes)             |
|-------------------------------------------------------|
|              LabelProg        (32 bytes)              |
|-------------------------------------------------------|
|              ProgVersion      (32 bytes)              |
|-------------------------------------------------------|
|              ProgDate         (32 bytes)              |
|-------------------------------------------------------|
:=======================================================:


Id: 32 byte Bacula identifier "Bacula 1.0 immortal\n"

(old version also recognized:)
Id: 32 byte Bacula identifier "Bacula 0.9 mortal\n"

LabelType (Saved in the FileIndex of the Header record).
    PRE_LABEL -1    Volume label on unwritten tape
    VOL_LABEL -2    Volume label after tape written
    EOM_LABEL -3    Label at EOM (not currently implemented)
    SOS_LABEL -4    Start of Session label (format given below)
    EOS_LABEL -5    End of Session label (format given below)


VerNum: 11

label_date: Julian day tape labeled
label_time: Julian time tape labeled

write_date: Julian date tape first used (data written)
write_time: Julian time tape first used (data written)

VolName: "Physical" Volume name

PrevVolName: The VolName of the previous tape (if this tape is
             a continuation of the previous one).
```

PoolName: Pool Name

PoolType: Pool Type

MediaType: Media Type

HostName: Name of host that is first writing the tape

LabelProg: Name of the program that labeled the tape

ProgVersion: Version of the label program

ProgDate: Date Label program built

Session Label

Id	(32 bytes)	
VerNum	(uint32_t)	
JobId	(uint32_t)	
write_btime	(btime_t)	VerNum 11
*write_date	(float64_t)	VerNum 10
0	(float64_t)	VerNum 11
*write_time	(float64_t)	VerNum 10
PoolName	(128 bytes)	
PoolType	(128 bytes)	
JobName	(128 bytes)	
ClientName	(128 bytes)	
Job	(128 bytes)	
FileSetName	(128 bytes)	
JobType	(uint32_t)	
JobLevel	(uint32_t)	
FileSetMD5	(50 bytes)	VerNum 11

Additional fields in End Of Session Label

JobFiles	(uint32_t)	
JobBytes	(uint32_t)	
start_block	(uint32_t)	
end_block	(uint32_t)	
start_file	(uint32_t)	

```
|-----------------------------------------------------|
|              end_file         (uint32_t)            |
|-----------------------------------------------------|
|              JobErrors        (uint32_t)            |
|-----------------------------------------------------|
|              JobStatus        (uint32_t) VerNum 11  |
:=====================================================:

* => fields deprecated

Id: 32 byte Bacula Identifier "Bacula 1.0 immortal\n"


LabelType (in FileIndex field of Header):
    EOM_LABEL -3     Label at EOM
    SOS_LABEL -4     Start of Session label
    EOS_LABEL -5     End of Session label

VerNum: 11

JobId: JobId

write_btime: Bacula time/date this tape record written

write_date: Julian date tape this record written - deprecated
write_time: Julian time tape this record written - deprecated.

PoolName: Pool Name

PoolType: Pool Type

MediaType: Media Type

ClientName: Name of File daemon or Client writing this session
            Not used for EOM_LABEL.
```

Unix File Attributes

The Unix File Attributes packet consists of the following:

<File-Index> <Type> <Filename>@<File-Attributes>@<Link> @<Extended-Attributes@>

where

@
: represents a byte containing a binary zero.

FileIndex
: is the sequential file index starting from one assigned by the File daemon.

Type
: is one of the following:

Filename
: is the fully qualified filename.

File-Attributes
: consists of the 13 fields of the stat() buffer in ASCII base64 format separated by spaces.

These fields and their meanings are shown below. This stat() packet is in Unix format, and MUST be provided (constructed) for ALL systems.

Link

when the FT code is FT_LNK or FT_LNKSAVED, the item in question is a Unix link, and this field contains the fully qualified link name. When the FT code is not FT_LNK or FT_LNKSAVED, this field is null.

Extended−Attributes

The exact format of this field is operating system dependent. It contains additional or extended attributes of a system dependent nature. Currently, this field is used only on WIN32 systems where it contains a ASCII base64 representation of the WIN32_FILE_ATTRIBUTE_DATA structure as defined by Windows. The fields in the base64 representation of this structure are like the File−Attributes separated by spaces.

The File−attributes consist of the following:

Field No.	Stat Name	Unix	Win98/NT	MacOS
1	st_dev	Device number of filesystem	Drive number	vRefNum
2	st_ino	Inode number	Always 0	fileID/dirID
3	st_mode	File mode	File mode	777 dirs/apps; 666 docs; 444 locked docs
4	st_nlink	Number of links to the file	Number of link (only on NTFS)	Always 1
5	st_uid	Owner ID	Always 0	Always 0
6	st_gid	Group ID	Always 0	Always 0
7	st_rdev	Device ID for special files	Drive No.	Always 0
8	st_size	File size in bytes	File size in bytes	Data fork file size in bytes
9	st_blksize	Preferred block size	Always 0	Preferred block size
10	st_blocks	Number of blocks allocated	Always 0	Number of blocks allocated
11	st_atime	Last access time since epoch	Last access time since epoch	Last access time −66 years
12	st_mtime	Last modify time since epoch	Last modify time since epoch	Last access time −66 years
13	st_ctime	Inode change time since epoch	File create time since epoch	File create time −66 years

Bacula 1.29 User's Guide Chapter

  
Storage Media Output Format Index Bacula TCP/IP Network Protocol

Bacula Memory Management

General

This document describes the memory management routines that are used in **Bacula** and is meant to be a technical discussion for developers rather than part of the user manual.

Since **Bacula** may be called upon to handle filenames of varying and more or less arbitrary length, special attention needs to be used in the code to ensure that memory buffers are sufficiently large. There are four possibilities for memory usage within **Bacula**. Each will be described in turn. They are:

- Statically allocated memory.
- Dynamically allocated memory using malloc() and free().
- Non−pooled memory.
- Pooled memory.

Statically Allocated Memory

Statically allocated memory is of the form:

```
char buffer[MAXSTRING];
```

The use of this kind of memory is discouraged except when you are 100% sure that the strings to be used will be of a fixed length. One example of where this is appropriate is for **Bacula** resource names, which are currently limited to 127 characters (MAX_NAME_LENGTH). Although this maximum size may change, particularly to accommodate Unicode, it will remain a relatively small value.

Dynamically Allocated Memory

Dynamically allocated memory is obtained using the standard malloc() routines. As in:

```
char *buf;
buf = malloc(256);
```

This kind of memory can be released with:

```
free(buf);
```

It is recommended to use this kind of memory only when you are sure that you know the memory size needed and the memory will be used for short periods of time −− that is it would not be appropriate to use statically allocated memory. An example might be to obtain a large memory buffer for reading and writing files. When **SmartAlloc** is enabled, the memory obtained by malloc() will automatically be checked for buffer overwrite (overflow) during the free() call, and all malloc'ed memory that is not released prior to termination of the program will be reported as Orphaned memory.

Pooled and Non−pooled Memory

In order to facility the handling of arbitrary length filenames and to efficiently handle a high volume of dynamic memory usage, we have implemented routines between the C code and the malloc routines. The first is called "Pooled" memory, and is memory, which once allocated and then released, is not returned to the system memory pool, but rather retained in a **Bacula** memory pool. The next request to acquire pooled memory will return any free memory block. In addition, each memory block has its current size associated with the block allowing for easy checking if the buffer is of sufficient size. This kind of memory would normally be used in high volume situations (lots of malloc()s and free()s) where the buffer length may have to frequently change to adapt to varying filename lengths.

The non−pooled memory is handled by routines similar to those used for pooled memory, allowing for easy size checking. However, non−pooled memory is returned to the system rather than being saved in the **Bacula** pool. This kind of memory would normally be used in low volume situations (few malloc()s and free()s), but where the size of the buffer might have to be adjusted frequently.

Types of Memory Pool

Currently there are three memory pool types:

- PM_NOPOOL −− non−pooled memory.
- PM_FNAME −− a filename pool.
- PM_MESSAGE −− a message buffer pool.
- PM_EMSG −− error message buffer pool.

Getting Memory

To get memory, one uses:

```
void *get_pool_memory(pool);
```

where **pool** is one of the above mentioned pool names. The size of the memory returned will be determined by the system to be most appropriate for the application.

If you wish non−pooled memory, you may alternatively call:

```
void *get_memory(size_t size);
```

The buffer length will be set to the size specified, and it will be assigned to the PM_NOPOOL pool (no pooling).

Releasing Memory

To free memory acquired by either of the above two calls, use:

```
void free_pool_memory(void *buffer);
```

where buffer is the memory buffer returned when the memory was acquired. If the memory was originally allocated as type PM_NOPOOL, it will be released to the system, otherwise, it will be

placed on the appropriate **Bacula** memory pool free chain to be used in a subsequent call for memory from that pool.

Determining the Memory Size

To determine the memory buffer size, use:

```
size_t sizeof_pool_memory(void *buffer);
```

Resizing Pool Memory

To resize pool memory, use:

```
void *realloc_pool_memory(void *buffer);
```

The buffer will be reallocated, and the contents of the original buffer will be preserved, but the address of the buffer may change.

Automatic Size Adjustment

To have the system check and if necessary adjust the size of your pooled memory buffer, use:

```
void *check_pool_memory_size(void *buffer, size_t new-size);
```

where **new-size** is the buffer length needed. Note, if the buffer is already equal to or larger than **new-size** no buffer size change will occur. However, if a buffer size change is needed, the original contents of the buffer will be preserved, but the buffer address may change. Many of the low level **Bacula** subroutines expect to be passed a pool memory buffer and use this call to ensure the buffer they use is sufficiently large.

Releasing All Pooled Memory

In order to avoid orphaned buffer error messages when terminating the program, use:

```
void close_memory_pool();
```

to free all unused memory retained in the **Bacula** memory pool. Note, any memory not returned to the pool via free_pool_memory() will not be released by this call.

Pooled Memory Statistics

For debugging purposes and performance tuning, the following call will print the current memory pool statistics:

```
void print_memory_pool_stats();
```

an example output is:

```
Pool  Maxsize  Maxused  Inuse
   0      256        0      0
   1      256        1      0
   2      256        1      0
```


Storage Media Output Format


Index


Bacula TCP/IP Network Protocol


Bacula Memory Management


Index


MD5 Algorithm

TCP/IP Network Protocol

General

This document describes the TCP/IP protocol used by Bacula to communicate between the various daemons and services. The definitive definition of the protocol can be found in src/lib/bnet.c and src/lib/netserver.c.

At the lowest level all packet transfers are done with read() and write() requests on system sockets. Pipes are not used as they are considered unreliable for large serial data transfers between various hosts.

Using the routines described below (bnet_open, bnet_write, bnet_recv, and bnet_close) guarantees that the number of bytes you write into the socket will be received as a single record on the other end regardless of how many low level write() and read() calls are needed. All data transferred are considered to be binary data.

bnet and Threads

These bnet routines work fine in a threaded environment. However, they assume that there is only one reader or writer on the socket at any time. It is highly recommended that only a single thread access any BSOCK packet. The exception to this rule is when the socket is first opened and it is waiting for a job to start. The wait in the Storage daemon is done in one thread and then passed to another thread.

If you envision having two threads using the same BSOCK, think twice, then you must implement some locking mechanism. It would not be appropriate to put locks inside the bnet subroutines.

bnet_open

To establish a connection to a server, use the subroutine:

BSOCK *bnet_open(char *host, char *service, int port)

bnet_open(), if successful, returns the Bacula sock descriptor pointer to be used in subsequent bnet_send() and bnet_read() requests. If not successful, bnet_open() returns a NULL.

bnet_send

To send a packet, one uses the subroutine:

int bnet_send(BSOCK *sock)

This routine is equivalent to a write() except that it handles the low level details. The data to be sent is expected to be in sock−>msg and be sock−>msglen bytes. To send a packet, bnet_send() first writes four bytes in network byte order than indicate the size of the following data packet. It returns:

```
 * Returns number of bytes sent
 * Returns −1 on error
```

bnet_fsend

This form uses:

int bnet_fsend(BSOCK *sock, char *format, ...)

and it allows you to send a formatted messages somewhat like fprintf().

bnet_recv

To read a packet, one uses the subroutine:

int bnet_recv(BSOCK *sock)

This routine is similar to a read() except that it handles the low level details. bnet_read() first reads packet length that follows as four bytes in network byte order. The data is read into sock−>msg and is sock−>msglen bytes. If the sock−>msg is not large enough, bnet_recv() realloc() the buffer. It will return an error (−2) if maxbytes is less than the record size sent. It returns:

```
 * Returns number of bytes read
 * Returns 0 on end of file
 * Returns −1 on hard end of file (i.e. network connection close)
 * Returns −2 on error
```

It should be noted that bnet_recv() is a blocking read.

bnet_sig

To send a "signal" from one daemon to another, one uses the subroutine:

int bnet_sig(BSOCK *sock, SIGNAL)

where SIGNAL is one of the following:

1. BNET_EOF − deprecated use BNET_EOD
2. BNET_EOD − End of data stream, new data may follow
3. BNET_EOD_POLL − End of data and poll all in one
4. BNET_STATUS − Request full status
5. BNET_TERMINATE − Conversation terminated, doing close()
6. BNET_POLL − Poll request, I'm hanging on a read
7. BNET_HEARTBEAT − Heartbeat Response requested
8. BNET_HB_RESPONSE − Only response permitted to HB
9. BNET_PROMPT − Prompt for UA

bnet_strerror

Returns a formated string corresponding to the last error that occurred.

bnet_close

The connection with the server remains open until closed by the subroutine:

void net_close(BSOCK *sock)

Becoming a Server

The bnet_open() and bnet_close() routines described above are used on the client side to establish a connection and terminate a connection with the server. To become a server (i.e. wait for a connection from a client), use the routine:

void net_server(int port, void handle_client_request())

where **port** is the port on which the client will listen, and **handle_client_request()** is the subroutine to be called when a client makes a valid connection. This routine fork()s for each client request and when the handle_client_request() returns, the child process terminates. Thus, a number of client requests can be processed simultaneously (currently set at 5). If multiple clients connect and there is a need to coordinate their data, the user must supply some form of communication and/or synchronization such as shared memory and semaphores.

Higher Level Conventions

Within Bacula, we have established the convention that any time a single record is passed, it is sent with bnet_send() and read with bnet_recv(). Thus the normal exchange between the server (S) and the client (C) are:

```
S: wait for connection            C: attempt connection
S: accept connection              C: bnet_send() send request
S: bnet_recv() wait for request
S: act on request
S: bnet_send() send ack            C: bnet_recv() wait for ack
```

Thus a single command is sent, acted upon by the server, and then acknowledged.

In certain cases, such as the transfer of the data for a file, all the information or data cannot be sent in a single packet. In this case, the convention is that the client will send a command to the server, who knows that more than one packet will be returned. In this case, the server will enter a loop:

```
while ((n=bnet_recv(bsock)) > 0) {
   act on request
}
if (n < 0)
   error
```

The client will perform the following:

```
bnet_send(bsock);
bnet_send(bsock);
...
bnet_sig(bsock, BNET_EOD);
```

Thus the client will send multiple packets and signal to the server when all the packets have been sent by sending a zero length record.


Bacula Memory Management

Index

MD5 Algorithm


Bacula TCP/IP Network Protocol

Index

Smart Memory Allocation

Command Line Message Digest Utility

This page describes **md5**, a command line utility usable on either Unix or MS−DOS/Windows, which generates and verifies message digests (digital signatures) using the MD5 algorithm. This program can be useful when developing shell scripts or Perl programs for software installation, file comparison, and detection of file corruption and tampering.

NAME

md5 − generate / check MD5 message digest

SYNOPSIS

md5 [**−c***signature*] [**−u**] [**−d***input_text* | *infile*] [*outfile*]

DESCRIPTION

A *message digest* is a compact digital signature for an arbitrarily long stream of binary data. An ideal message digest algorithm would never generate the same signature for two different sets of input, but achieving such theoretical perfection would require a message digest as long as the input file. Practical message digest algorithms compromise in favour of a digital signature of modest size created with an algorithm designed to make preparation of input text with a given signature computationally infeasible. Message digest algorithms have much in common with techniques used in encryption, but to a different end; verification that data have not been altered since the signature was published.

Many older programs requiring digital signatures employed 16 or 32 bit *cyclical redundancy codes* (CRC) originally developed to verify correct transmission in data communication protocols, but these short codes, while adequate to detect the kind of transmission errors for which they were intended, are insufficiently secure for applications such as electronic commerce and verification of security related software distributions.

The most commonly used present−day message digest algorithm is the 128 bit MD5 algorithm, developed by Ron Rivest of the MIT Laboratory for Computer Science and RSA Data Security, Inc. The algorithm, with a reference implementation, was published as Internet RFC 1321 in April 1992, and was placed into the public domain at that time. Message digest algorithms such as MD5 are not deemed "encryption technology" and are not subject to the export controls some governments impose on other data security products. (Obviously, the responsibility for obeying the laws in the jurisdiction in which you reside is entirely your own, but many common Web and Mail utilities use MD5, and I am unaware of any restrictions on their distribution and use.)

The MD5 algorithm has been implemented in numerous computer languages including C, Perl, and Java; if you're writing a program in such a language, track down a suitable subroutine and incorporate it into your program. The program described on this page is a *command line* implementation of MD5, intended for use in shell scripts and Perl programs (it is much faster than computing an MD5 signature directly in Perl). This **md5** program was originally developed as part of a suite of tools intended to monitor large collections of files (for

example, the contents of a Web site) to detect corruption of files and inadvertent (or perhaps malicious) changes. That task is now best accomplished with more comprehensive packages such as Tripwire, but the command line **md5** component continues to prove useful for verifying correct delivery and installation of software packages, comparing the contents of two different systems, and checking for changes in specific files.

OPTIONS

*−**c**signature*
: Computes the signature of the specified *infile* or the string supplied by the **−d** option and compares it against the specified *signature*. If the two signatures match, the exit status will be zero, otherwise the exit status will be 1. No signature is written to *outfile* or standard output; only the exit status is set. The signature to be checked must be specified as 32 hexadecimal digits.

*−**d**input_text*
: A signature is computed for the given *input_text* (which must be quoted if it contains white space characters) instead of input from *infile* or standard input. If input is specified with the **−d** option, no *infile* should be specified.

−u
: Print how−to−call information.

FILES

If no *infile* or **−d** option is specified or *infile* is a single "−", **md5** reads from standard input; if no *outfile* is given, or *outfile* is a single "−", output is sent to standard output. Input and output are processed strictly serially; consequently **md5** may be used in pipelines.

BUGS

The mechanism used to set standard input to binary mode may be specific to Microsoft C; if you rebuild the DOS/Windows version of the program from source using another compiler, be sure to verify binary files work properly when read via redirection or a pipe.

This program has not been tested on a machine on which `int` and/or `long` are longer than 32 bits.

 **Download md5.zip (Zipped archive)**

The program is provided as md5.zip, a Zipped archive containing an ready−to−run Win32 command−line executable program, `md5.exe` (compiled using Microsoft Visual C++ 5.0), and in source code form along with a `Makefile` to build the program under Unix.

SEE ALSO

sum(1)

EXIT STATUS

md5 returns status 0 if processing was completed without errors, 1 if the **−c** option was specified and the given signature does not match that of the input, and 2 if processing could not be performed at all due, for example, to a nonexistent input file.

COPYING

ACKNOWLEDGEMENTS

The MD5 algorithm was developed by Ron Rivest. The public domain C language implementation used in this program was written by Colin Plumb in 1993.

---

by John Walker
January 6th, MIM


Bacula TCP/IP Network Protocol

Index

Smart Memory Allocation

Bacula 1.29 User's Guide Chapter


Smart Memory Allocation

Index

Smartalloc

Smart Memory Allocation With Orphaned Buffer Detection

Few things are as embarrassing as a program that leaks, yet few errors are so easy to commit or as difficult to track down in a large, complicated program as failure to release allocated memory. SMARTALLOC replaces the standard C library memory allocation functions with versions which keep track of buffer allocations and releases and report all orphaned buffers at the end of program execution. By including this package in your program during development and testing, you can identify code that loses buffers right when it's added and most easily fixed, rather than as part of a crisis debugging push when the problem is identified much later in the testing cycle (or even worse, when the code is in the hands of a customer). When program testing is complete, simply recompiling with different flags removes SMARTALLOC from your program, permitting it to run without speed or storage penalties.

In addition to detecting orphaned buffers, SMARTALLOC also helps to find other common problems in management of dynamic storage including storing before the start or beyond the end of an allocated buffer, referencing data through a pointer to a previously released buffer, attempting to release a buffer twice or releasing storage not obtained from the allocator, and assuming the initial contents of storage allocated by functions that do not guarantee a known value. SMARTALLOC's checking does not usually add a large amount of overhead to a program (except for programs which use `realloc()` extensively; see below). SMARTALLOC focuses on proper storage management rather than internal consistency of the heap as checked by the malloc_debug facility available on some systems. SMARTALLOC does not conflict with malloc_debug and both may be used together, if you wish. SMARTALLOC makes no assumptions regarding the internal structure of the heap and thus should be compatible with any C language implementation of the standard memory allocation functions.

Installing SMARTALLOC

SMARTALLOC is provided as a Zipped archive, smartall.zip; see the download instructions below.

To install SMARTALLOC in your program, simply add the statement:

```
#include "smartall.h"
```

to every C program file which calls any of the memory allocation functions (`malloc`, `calloc`, `free`, etc.). SMARTALLOC must be used for all memory allocation with a program, so it's best to add the #include of "smartall.h" to the omnibus include file for your entire program, if you have such a thing. Next, define the symbol SMARTALLOC in the compilation before the inclusion of smartall.h. I usually do this by having my Makefile add the "`-DSMARTALLOC`" option to the C compiler for non−production builds. You can define the symbol manually, if you prefer, by adding the statement:

```
#define SMARTALLOC
```

before the #include of "smartall.h".

At the point where your program is all done and ready to relinquish control to the operating system, add the call:

```
sm_dump(datadump);
```

where *datadump* specifies whether the contents of orphaned buffers are to be dumped in addition printing to their size and place of allocation. The data are dumped only if *datadump* is nonzero, so most programs will normally use "`sm_dump(0);`". If a mysterious orphaned buffer appears that can't be identified from the information this prints about it, replace the statement with "`sm_dump(1);`". Usually the dump of the buffer's data will furnish the additional clues you need to excavate and extirpate the elusive error that left the buffer allocated.

Finally, add the files "smartall.h" and "smartall.c" from this release to your source directory, make dependencies, and linker input. You needn't make inclusion of smartall.c in your link optional; if compiled with SMARTALLOC not defined it generates no code, so you may always include it knowing it will waste no storage in production builds. Now when you run your program, if it leaves any buffers around when it's done, each will be reported by `sm_dump()` on stderr as follows:

```
Orphaned buffer:    120 bytes allocated at line 50 of gutshot.c
```

Squelching a SMARTALLOC

Usually, when you first install SMARTALLOC in an existing program you'll find it nattering about lots of orphaned buffers. Some of these turn out to be legitimate errors, but some are storage allocated during program initialisation that, while dynamically allocated, is logically static storage not intended to be released. Of course, you can get rid of the complaints about these buffers by adding code to release them, but by doing so you're adding unnecessary complexity and code size to your program just to silence the nattering of a SMARTALLOC, so an escape hatch is provided to eliminate the need to release these buffers.

Normally all storage allocated with the functions `malloc()`, `calloc()`, and `realloc()` is monitored by SMARTALLOC. If you make the function call:

```
sm_static(1);
```

you declare that subsequent storage allocated by `malloc()`, `calloc()`, and `realloc()` should not be considered orphaned if found to be allocated when `sm_dump()` is called. I use a call on "`sm_static(1);`" before I allocate things like program configuration tables so I don't have to add code to release them at end of program time. After allocating unmonitored data this way, be sure to add a call to:

```
sm_static(0);
```

to resume normal monitoring of buffer allocations. Buffers allocated while `sm_static(1)` is in effect are not checked for having been orphaned but all the other safeguards provided by SMARTALLOC remain in effect. You may release such buffers, if you like; but you don't have to.

Living with Libraries

Some library functions for which source code is unavailable may gratuitously allocate and return buffers that contain their results, or require you to pass them buffers which they subsequently release. If you have source code for the library, by far the best approach is to simply install SMARTALLOC in it, particularly since this kind of ill−structured dynamic storage management is the source of so many storage leaks. Without source code, however, there's no option but to provide a way to bypass SMARTALLOC for the buffers the library allocates and/or releases with the standard system functions.

For each function *xxx* redefined by SMARTALLOC, a corresponding routine named "`actually`*xxx*" is furnished which provides direct access to the underlying system function, as follows:

Standard function	**Direct access function**
`malloc(`*size*`)`	`actuallymalloc(`*size*`)`
`calloc(`*nelem*`, `*elsize*`)`	`actuallycalloc(`*nelem*`, `*elsize*`)`
`realloc(`*ptr*`, `*size*`)`	`actuallyrealloc(`*ptr*`, `*size*`)`
`free(`*ptr*`)`	`actuallyfree(`*ptr*`)`

For example, suppose there exists a system library function named "`getimage()`" which reads a raster image file and returns the address of a buffer containing it. Since the library routine allocates the image directly with `malloc()`, you can't use SMARTALLOC's `free()`, as that call expects information placed in the buffer by SMARTALLOC's special version of `malloc()`, and hence would report an error. To release the buffer you should call `actuallyfree()`, as in this code fragment:

```
struct image *ibuf = getimage("ratpack.img");
display_on_screen(ibuf);
actuallyfree(ibuf);
```

Conversely, suppose we are to call a library function, "`putimage()`", which writes an image buffer into a file and then releases the buffer with `free()`. Since the system `free()` is being called, we can't pass a buffer allocated by SMARTALLOC's allocation routines, as it contains special information that the system `free()` doesn't expect to be there. The following code uses `actuallymalloc()` to obtain the buffer passed to such a routine.

```
struct image *obuf =
   (struct image *) actuallymalloc(sizeof(struct image));
dump_screen_to_image(obuf);
putimage("scrdump.img", obuf);  /* putimage() releases obuf */
```

It's unlikely you'll need any of the "actually" calls except under very odd circumstances (in four products and three years, I've only needed them once), but they're there for the rare occasions that demand them. Don't use them to subvert the error checking of SMARTALLOC; if you want to disable orphaned buffer detection, use the `sm_static(1)` mechanism described above. That way you don't forfeit all the other advantages of SMARTALLOC as you do when using `actuallymalloc()` and `actuallyfree()`.

SMARTALLOC Details

When you include "smartall.h" and define SMARTALLOC, the following standard system library functions are redefined with the instead. (For details of the redefinitions, please refer to smartall.h.)

```
void *malloc(size_t size)
void *calloc(size_t nelem, size_t elsize)
void *realloc(void *ptr, size_t size)
void free(void *ptr)
void cfree(void *ptr)
```

`cfree()` is a historical artifact identical to `free()`.

In addition to allocating storage in the same way as the standard library functions, the SMARTALLOC versions expand the buffers they allocate to include information that identifies where each buffer was allocated and to chain all allocated buffers together. When a buffer is released, it is removed from the allocated buffer chain. A call on `sm_dump()` is able, by scanning the chain of allocated buffers, to find all orphaned buffers. Buffers allocated while `sm_static(1)` is in effect are specially flagged so that, despite appearing on the allocated buffer chain, `sm_dump()` will not deem them orphans.

When a buffer is allocated by `malloc()` or expanded with `realloc()`, all bytes of newly allocated storage are set to the hexadecimal value 0x55 (alternating one and zero bits). Note that for `realloc()` this applies only to the bytes added at the end of buffer; the original contents of the buffer are not modified. Initialising allocated storage to a distinctive nonzero pattern is intended to catch code that erroneously assumes newly allocated buffers are cleared to zero; in fact their contents are random. The `calloc()` function, defined as returning a buffer cleared to zero, continues to zero its buffers under SMARTALLOC.

Buffers obtained with the SMARTALLOC functions contain a special sentinel byte at the end of the user data area. This byte is set to a special key value based upon the buffer's memory address. When the buffer is released, the key is tested and if it has been overwritten an assertion in the `free` function will fail. This catches incorrect program code that stores beyond the storage allocated for the buffer. At `free()` time the queue links are also validated and an assertion failure will occur if the program has destroyed them by storing before the start of the allocated storage.

In addition, when a buffer is released with `free()`, its contents are immediately destroyed by overwriting them with the hexadecimal pattern 0xAA (alternating bits, the one's complement of the initial value pattern). This will usually trip up code that keeps a pointer to a buffer that's been freed and later attempts to reference data within the released buffer. Incredibly, this is *legal* in the standard Unix memory allocation package, which permits programs to free() buffers, then raise them from the grave with `realloc()`. Such program "logic" should be fixed, not accommodated, and SMARTALLOC brooks no such Lazarus buffer" nonsense.

Some C libraries allow a zero size argument in calls to `malloc()`. Since this is far more likely to indicate a program error than a defensible programming stratagem, SMARTALLOC disallows it with an assertion.

When the standard library `realloc()` function is called to expand a buffer, it attempts to

expand the buffer in place if possible, moving it only if necessary. Because SMARTALLOC must place its own private storage in the buffer and also to aid in error detection, its version of `realloc()` always moves and copies the buffer except in the trivial case where the size of the buffer is not being changed. By forcing the buffer to move on every call and destroying the contents of the old buffer when it is released, SMARTALLOC traps programs which keep pointers into a buffer across a call on `realloc()` which may move it. This strategy may prove very costly to programs which make extensive use of `realloc()`. If this proves to be a problem, such programs may wish to use `actuallymalloc()`, `actuallyrealloc()`, and `actuallyfree()` for such frequently−adjusted buffers, trading error detection for performance. Although not specified in the System V Interface Definition, many C library implementations of `realloc()` permit an old buffer argument of NULL, causing `realloc()` to allocate a new buffer. The SMARTALLOC version permits this.

When SMARTALLOC is Disabled

When SMARTALLOC is disabled by compiling a program with the symbol SMARTALLOC not defined, calls on the functions otherwise redefined by SMARTALLOC go directly to the system functions. In addition, compile−time definitions translate calls on the "`actually...()`" functions into the corresponding library calls; "`actuallymalloc(100)`", for example, compiles into "`malloc(100)`". The two special SMARTALLOC functions, `sm_dump()` and `sm_static()`, are defined to generate no code (hence the null statement). Finally, if SMARTALLOC is not defined, compilation of the file smartall.c generates no code or data at all, effectively removing it from the program even if named in the link instructions.

Thus, except for unusual circumstances, a program that works with SMARTALLOC defined for testing should require no changes when built without it for production release.

The `alloc()` Function

Many programs I've worked on use very few direct calls to `malloc()`, using the identically declared `alloc()` function instead. Alloc detects out−of−memory conditions and aborts, removing the need for error checking on every call of `malloc()` (and the temptation to skip checking for out−of−memory).

As a convenience, SMARTALLOC supplies a compatible version of `alloc()` in the file alloc.c, with its definition in the file alloc.h. This version of `alloc()` is sensitive to the definition of SMARTALLOC and cooperates with SMARTALLOC's orphaned buffer detection. In addition, when SMARTALLOC is defined and `alloc()` detects an out of memory condition, it takes advantage of the SMARTALLOC diagnostic information to identify the file and line number of the call on `alloc()` that failed.

Overlays and Underhandedness

String constants in the C language are considered to be static arrays of characters accessed through a pointer constant. The arrays are potentially writable even though their pointer is a constant. SMARTALLOC uses the compile−time definition `./smartall.wml` to obtain the name of the file in which a call on buffer allocation was performed. Rather than reserve space in a buffer to save this information, SMARTALLOC simply stores the pointer to the compiled−in text of the file name. This works fine as long as the program does not overlay its data among modules. If data are overlayed, the area of memory which contained the file name at the time it

was saved in the buffer may contain something else entirely when `sm_dump()` gets around to using the pointer to edit the file name which allocated the buffer.

If you want to use SMARTALLOC in a program with overlayed data, you'll have to modify smartall.c to either copy the file name to a fixed−length field added to the `abufhead` structure, or else allocate storage with `malloc()`, copy the file name there, and set the `abfname` pointer to that buffer, then remember to release the buffer in `sm_free`. Either of these approaches are wasteful of storage and time, and should be considered only if there is no alternative. Since most initial debugging is done in non−overlayed environments, the restrictions on SMARTALLOC with data overlaying may never prove a problem. Note that conventional overlaying of code, by far the most common form of overlaying, poses no problems for SMARTALLOC; you need only be concerned if you're using exotic tools for data overlaying on MS−DOS or other address−space−challenged systems.

Since a C language "constant" string can actually be written into, most C compilers generate a unique copy of each string used in a module, even if the same constant string appears many times. In modules that contain many calls on allocation functions, this results in substantial wasted storage for the strings that identify the file name. If your compiler permits optimisation of multiple occurrences of constant strings, enabling this mode will eliminate the overhead for these strings. Of course, it's up to you to make sure choosing this compiler mode won't wreak havoc on some other part of your program.

Test and Demonstration Program

A test and demonstration program, smtest.c, is supplied with SMARTALLOC. You can build this program with the Makefile included. Please refer to the comments in smtest.c and the Makefile for information on this program. If you're attempting to use SMARTALLOC on a new machine or with a new compiler or operating system, it's a wise first step to check it out with smtest first.

Invitation to the Hack

SMARTALLOC is not intended to be a panacea for storage management problems, nor is it universally applicable or effective; it's another weapon in the arsenal of the defensive professional programmer attempting to create reliable products. It represents the current state of evolution of expedient debug code which has been used in several commercial software products which have, collectively, sold more than third of a million copies in the retail market, and can be expected to continue to develop through time as it is applied to ever more demanding projects.

The version of SMARTALLOC here has been tested on a Sun SPARCStation, Silicon Graphics Indigo², and on MS−DOS using both Borland and Microsoft C. Moving from compiler to compiler requires the usual small changes to resolve disputes about prototyping of functions, whether the type returned by buffer allocation is `char *` or `void *`, and so forth, but following those changes it works in a variety of environments. I hope you'll find SMARTALLOC as useful for your projects as I've found it in mine.

 **Download smartall.zip (Zipped archive)**

SMARTALLOC is provided as smartall.zip, a Zipped archive containing source code, documentation, and a `Makefile` to build the software under Unix.

Copying

> SMARTALLOC is in the public domain. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, without any conditions or restrictions. This software is provided "as is" without express or implied warranty.

---

by John Walker
October 30th, 1998


Smart Memory Allocation


Index
