Bacula Enterprise Technical Reference

Bacula Systems Documentation

Contents

Contents

The following chapter aims at presenting the reader with Bacula technical content.

1 Director

The following chapter aims at presenting the reader with Bacula technical content regarding the Director.

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

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

Director Resource Example

```
Director {
    Name = bacula-dir
    WorkingDirectory = "/opt/bacula/working"
    Password = UA_password
    PidDirectory = "/opt/bacula/working"
    QueryFile = "/opt/bacula/query.sql"
    Messages = Standard
}
```

Read more:

1.1 Director Resource Types

Director resource types may be the following:

Note: Everything revolves around a job and is tied to a job in one way or another.

- *Director* – to define the Director's name and its access password used for authenticating the Console program. Only a single Director resource definition may appear in the Director's configuration file. If you have either

/dev/random or **bc** on your machine, **Bacula** will generate a random password during the configuration process, otherwise it will be left blank.

- *Job* – to define the backup/restore Jobs and to tie together the Client, FileSet and Schedule resources to be used for each Job. Normally, you will have Jobs of different names corresponding to each client (i.e., one Job per client, but a different one with a different name for each client).
- *JobDefs*– optional resource for providing defaults for Job resources.
- *Schedule* – to define when a Job is to be automatically run by **Bacula**'s internal scheduler. You may have any number of Schedules, but each job will reference only one.
- *FileSet* – to define the set of files to be backed up with each job. You may have any number of FileSets but each Job will reference only one.
- *Client* – to define what Client is to be backed up. You will generally have multiple Client definitions. Each Job will reference only a single client.
- *Storage/Autochanger* – to define on what physical device the Volumes should be mounted. You may have one or more Storage definitions.
- *Pool* – to define the pool of Volumes that can be used for a particular Job. If there is a large number of clients or volumes, you may want to have multiple Pools. Pools allow you to restrict a Job (or a Client) to use only a particular set of Volumes.
- *Catalog* – to define in what database to keep the list of files and the Volume names where they are backed up. Most people only use a single catalog. However, if you want to scale the Director to many clients, multiple catalogs can be helpful. Multiple catalogs require a bit more management because in general you must know what catalog contains what data. Currently, all Pools are defined in each catalog. This restriction will be removed in a later release.
- *Messages* – to define where to error and information messages are to be sent or logged. You may define multiple different message resources and hence direct particular classes of messages to different users or locations (files, etc.).
- *Console* – to define how the administrator or user can interact with the Director.
- *Counter* – to define a counter variable that can be accessed by variable expansion used for creating Volume labels.

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 a future design may contain multiple Directors to maintain index and media database redundancy.

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

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

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

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

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

MalwareDatabaseCommand = <Command> Specifies the command used to update the Malware database. The default Malware database handler uses abuse.ch (get_malware_abuse.ch). The database is synchronized automatically when an update is available.

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

Plugin Directory = <Directory> Each daemon (DIR, FD, SD) has a **Plugin Directory** directive that may be added to the daemon definition resource. The directory takes a quoted string argument, which is the name of the directory in which the daemon can find the Bacula plugins. If this directive is not specified, Bacula will not load any plugins. Since each plugin has a distinctive name, all the daemons can share the same plugin directory.

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

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

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

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

By default, Bacula will use the working directory to store the process id file. This directive is required.

Scripts Directory = <Directory> This directive is optional and, if defined, specifies a directory in which the Director and the Storage daemon will look for many of the scripts that it needs to use during particular operations such as starting/stopping, the **mtx-changer** script, tape alerts, as well as catalog updates. This directory may be shared by other **Bacula** daemons. Standard shell expansion of the directory is done when the configuration file is read so that if you wish to use values such as **$HOME** they will be properly expanded.

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

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Director to set a keepalive interval (heartbeat) in seconds on each of the sockets it opens for the Client resource. This value will override any specified at the Director level. It is implemented only on systems (Linux, ...) that provide the **setsockopt** `TCP_KEEPIDLE` function. The default value is **300s**.

Maximum Concurrent Jobs = <number> where **<number>** is the maximum number of total Director Jobs that should run concurrently. The default is set to **20**, but you may set it to a larger number. Every valid connection to any daemon (Director, File daemon, or Storage daemon) results in a Job. This includes connections from `bconsole`. Thus the number of concurrent Jobs must, in general, be greater than the maximum number of Jobs that you wish to actually run.

In general, increasing the number of Concurrent Jobs increases the total throughput of **Bacula**, because the simultaneous Jobs can all feed data to the Storage daemon and to the Catalog at the same time. However, keep in mind, that the Volume format becomes more complicated with multiple simultaneous jobs, consequently, restores may take longer if must sort through interleaved volume blocks from multiple simultaneous jobs. Though not normally necessary, this can be avoided by having each simultaneous job write to a different volume or by using data spooling, which will first spool

the data to disk simultaneously, then write one spool file at a time to the volume thus avoiding excessive interleaving of the different job blocks.

Maximum Reload Requests = <number> defines how many reload requests can be queued, waiting for resources that are locked while Jobs are active.

```
Director {
    Name = localhost-dir
    Maximum Reload Requests = 64
    ...
}
```

The default value is 32. Increasing this value should be considered carefully, as the operational consequences can be considerable.

FD Connect Timeout = <time> where **<time>** is the time that the Director should continue attempting to contact the File daemon to start a job, and after which the Director will cancel the job. The default is **3 minutes**.

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

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

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

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

Note: If you use the DirAddresses directive, you must not use either a DirPort or a DirAddress directive in the same resource.

DirPort = <port-number> Specify the port (a positive integer) on which the Director daemon will listen for **Bacula** Console connections. This same port number must be specified in the Director resource of the Console configuration

file. The default is **9101**, so normally this directive need not be specified. This directive should not be used if you specify the DirAddresses (plural) directive.

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

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

EventsRetention = <time> The **Events Retention** directive defines the length of time that **Bacula** will keep events records in the Catalog database. When this time period expires, and if the user runs the command **prune events** command, **Bacula** will prune (remove) Events records that are older than the specified period.

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

The default is **1 month**.

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

Theses statistics records aren't used for restore purpose, but mainly for capacity planning, billings, etc.

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

The default is **5 years**.

AutoPrune = <yes|no> Normally, pruning of Files from the Catalog is specified on a Client by Client basis in the resource with the **AutoPrune** directive. It is also possible to overwrite the Client settings in the resource used by jobs, with the **AutoPrune**, **PruneFiles** and **PruneJobs** directives.

If this directive is specified (not normally) and the value is **no**, it will override the value specified in all the other resources. The default is **yes**.

If you set **AutoPrune=no**, pruning will not be done automatically, and your Catalog will grow in size each time you run a Job. Pruning affects only information in the catalog and not data stored in the backup archives (on Volumes). The **prune bconsole** command can be used to prune catalog records respecting the Client and/or the Pool **FileRetention**, **JobRetention** and **VolumeRetention** directives.

CustomerId = <string> where **<string>** is an identifier which can be used for support purpose.

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

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

MaximumReloadRequests = <number> Where **<number>** is the maximum number of **reload** command that can be queued while jobs are running. If the limit is reached, subsequent reloads will not be possible until some of the currently running jobs are finished. The default is set to **32** and is usually sufficient, even in large infrastructures.

FIPS Require = <yes|no> Require FIPS cryptographic module to start the daemon.

CommCompression = <yes|no> If the two **Bacula** components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes.

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is **enabled**. In the case that the compression is not effective, **Bacula** turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, **Bacula** reports **None** in the Job report.

Note: TLS Directives in the Director resource of bacula-dir.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh** . The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

See also:

Go to:

- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Job Resource

The Job resource defines a Job (Backup, Restore, etc.) that **Bacula** must perform. Each Job resource definition contains the name of a Client and a FileSet to backup, the Schedule for the Job, where the data are to be stored, and what media Pool can be used. In effect, each Job resource must specify What, Where, How, and When or FileSet, Storage, Backup/Restore/Level, and Schedule respectively.

Note: The FileSet must be specified for a restore job for historical reasons, but it is no longer used.

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

Bacula uses “Internal system jobs” and “Console connection jobs”. Those are for internal processes, among others the ones with “JobID = 0”, and should be considered as meaningful when logs are displayed.

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

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

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

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

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

Enabled = <yes|no> This directive allows you to enable or disable a resource. When the resource of the Job is disabled, the Job will no longer be scheduled and it will not be available in the list of Jobs to be run. To be able to use the Job you must **enable** it.

Tag = <string, string2, string3> The **Tag** directive specifies a list of tags to create when creating a new Job record. This directive is optional.

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

- **Backup** Run a backup Job. Normally you will have at least one Backup job for each client you want to save. Normally, unless you turn off cataloging, most all the important statistics and data concerning files backed up will be placed in the Catalog.
- **Restore** Run a restore Job. Normally, you will specify only one Restore job which acts as a sort of prototype that you will modify using the console program in order to perform restores. Although certain basic information from a Restore job is saved in the catalog, it is very minimal compared to the information stored for a Backup job – for example, no File database entries are generated since no Files are saved.

  Restore jobs cannot be automatically started by the scheduler as is the case for Backup, Verify and Admin jobs. To restore files, you must use the **restore** command in the console.
- **Verify** Run a **Verify** Job. In general, **Verify** jobs permit you to compare the contents of the catalog to the file system, or to what was backed up. In addition, to verifying that a tape that was written can be read, you can also use **Verify** as a sort of tripwire intrusion detection.
- **Admin** Run an **Admin** Job. Only Director’s runscripts will be executed. The Client is not involved in an Admin job, so features such as **Client Run Before Job** are not available. Although an Admin job is recorded in the catalog, very little data is saved. 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.
- **Migration** Run a **Migration** Job (similar to a backup job) that reads data that was previously backed up to a Volume and writes it to another Volume (see).

- **Copy** Run a **Copy** Job that essentially creates two identical copies of the same backup. The **Copy** process is essentially identical to the Migration feature with the exception that the Job that is copied is left unchanged (see).

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

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

- **Full** When the **Level** is set to **Full** all files in the FileSet whether or not they have changed will be backed up.
- **Incremental** When the Level is set to Incremental all files specified in the FileSet that have changed since the last successful backup of the the same Job using the same FileSet and Client, will be backed up. If the Director cannot find a previous valid Full backup then the job will be upgraded into a Full backup. When the Director looks for a valid backup record in the catalog database, it looks for a previous Job with:
  - The same Job name.
  - The same Client name.
  - The same FileSet (any change to the definition of the FileSet such as adding or deleting a file in the Include or Exclude sections constitutes a different FileSet.
  - The Job was a Full, Differential, or Incremental backup.
  - The Job terminated normally (i.e. did not fail or was not canceled).
  - The Job started no longer ago than **Max Full Interval**.

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

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

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

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

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

  However, to manage deleted files or directories changes in the catalog during an Incremental backup you can use **accurate** mode. This is quite memory consuming process. See Accurate mode for more details.
- **Differential** When the Level is set to Differential all files specified in the FileSet that have changed since the last successful Full backup of the same Job will be backed up. If the Director cannot find a valid previous Full backup for the same Job, FileSet, and Client, backup, then the Differential job will be upgraded into a Full backup. When the Director looks for a valid Full backup record in the catalog database, it looks for a previous Job with:
  - The same Job name.

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

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

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

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

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

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

Every once and a while, someone asks why we need Differential backups as long as Incremental backups pickup all changed files. There are possibly many answers to this question, but the one that is the most important for me is that a Differential backup effectively merges all the Incremental and Differential backups since the last Full backup into a single Differential backup. This has two effects:

1. It gives some redundancy since the old backups could be used if the merged backup cannot be read.
2. More importantly, it reduces the number of Volumes that are needed to do a restore effectively eliminating the need to read all the volumes on which the preceding Incremental and Differential backups since the last Full are done.

- **VirtualFull** When the backup Level is set to **VirtualFull**, **Bacula** will consolidate the previous Full backup plus the most recent Differential backup and any subsequent Incremental backups into a new Full backup. This new Full backup will then be considered as the most recent Full for any future Incremental or Differential backups. The VirtualFull backup is accomplished without contacting the client by reading the previous backup data and writing it to a volume in a different pool.

  Bacula's virtual backup feature is often called Synthetic Backup or Consolidation in other backup products.

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

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

- **InitCatalog** does a scan of the specified **FileSet** and stores the file attributes in the Catalog database. Since no file data is saved, you might ask why you would want to do this. It turns out to be a very simple and easy way to have a **Tripwire** like feature using **Bacula**. In other words, it allows you to save the state of a set of files defined by the 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 that contains the set of system files that should not change (e.g. /sbin, /boot, /lib, /bin, etc.). Normally, you run the **InitCatalog** level verify one time when your system is first setup, and then once again after each modification (upgrade) to your system. Thereafter, when your want to check the state of your system files, you use a **Verify level=Catalog**. This compares the results of your **InitCatalog** with the current state of the files.
- **Catalog** Compares the current state of the files against the state previously saved during an **InitCatalog**. Any discrepancies are reported. The items reported are determined by the **Verify** options specified on the **Include** directive in the specified (see the 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.

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

- **Data** Read back the data stored on volumes and check data attributes such as size and the checksum of all the files.

  To run the Verify job, it is possible to use the "jobid" parameter of the "run" command.

  Note: The current Verify Data implementation requires specifying the correct Storage resource in the Verify job. The Storage resource can be changed with the bconsole command line and with the menu.

  It is also possible to use the `accurate` option to check catalog records at the same time. When using a Verify job with `level=Data` and `accurate=yes` can replace the `level=VolumeToCatalog` option.

  To run a Verify Job with the `accurate` option, it is possible to set the option in the Job definition or set use the `accurate=yes` on the command line.

```
* run job=VerifyData level=Data jobid=10 accurate=yes
```

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

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

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

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

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

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

If the "accurate" option is not specified in the FileSet, Bacula will use by default the "mcs" options:

- m compare the modification time (st_mtime)
- c compare the change time (st_ctime)
- s compare the size

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

For 500.000 files, it will require approximately 64 Megabytes of RAM on your File Daemon to hold the required information.

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

Plugin Options = <Plugin-Command-Line> If you run a Verify Job with the level Data, it is possible to specify a Plugin command that will be used during the Job. For example, it can be used in conjunction with the Antivirus plugin. The directive can be overwritten from the run menu, or from the run command line with the **pluginoptions=** keyword.

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

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

If you use the **restore** command in the **bconsole** 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** directive, see Restoring Files with the Bootstrap File chapter of this manual.

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

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

If it 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.

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

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

For more details on using this file, see the The Bootstrap File chapter.

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

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

Base = <job-resource-name,> The Base directive permits to specify the list of jobs that will be used during Full backup as base. This directive is optional. See the Base Job chapter for more information.

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

Snapshot Retention = <time-period-specification> The Snapshot Retention directive defines the length of time that **Bacula** will keep Snapshots in the Catalog database and on the Client after the Snapshot creation. When this time period expires, and if using the **snapshot prune** command, **Bacula** will prune (remove) Snapshot records that are older than the specified Snapshot Retention period and will contact the FileDaemon to delete Snapshots from the system.

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

The default is **0 second** , Snapshots are deleted at the end of the backup. The Job **SnapshotRetention** directive overwrites the Client **SnapshotRetention** directive.

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

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

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

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

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

BackupsToKeep = <number> When this directive is present during a Virtual Full (it is ignored for other Job types), it will look for a Full backup that has more subsequent backups than the value specified. In the example below, the Job will simply terminate unless there is a Full back followed by at least 31 backups of either level Differential or Incremental.


```
Job {
    Name = "VFull"
    Type = Backup
    Level = VirtualFull
    Client = "my-fd"
    File Set = "FullSet"
    Accurate = Yes
    Backups To Keep = 30
}
```

Assuming that the last Full backup is followed by 32 Incremental backups, a Virtual Full will be run that consolidates the Full with the first two Incrementals that were run after the Full. The result is that you will end up with a Full followed by 30 Incremental backups.

Setting "BackupsToKeep = 0" will cause Bacula to run the Virtual Full if you have at least 1 subsequent incr/diff job after the Full level job and no incr/diff job is kept after the Virtual Full job has run. This is the default behavior if this directive is not set.

DeleteConsolidatedJobs = <yes/no> If set to **yes**, it will cause any old Job that is consolidated during a Virtual Full to be deleted. In the example above we saw that a Full plus one other job (either an Incremental or Differential) were consolidated into a new Full backup. The original Full plus the other Job consolidated will be deleted. The default value is **no**.

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

Storage = <storage-resource-name> The Storage directive defines the name of the storage services where you want to backup the FileSet data. For additional details, see the *Storage Resource Chapter*. The Storage resource may also be specified in the Job's Pool resource, in which case the value in the Pool resource overrides any value in the Job. This Storage resource definition is not required by either the Job resource or in the Pool, but it must be specified in one or the other, if not an error will result. Storage can be either specified by an single item or it can be specified as a comma separated list of storages to use according to StorageGroupPolicy. If some number of first storage daemons on the list are unavailable due to network problems, broken or unreachable for some other reason, Bacula will take first available one from the list (which is sorted according to the policy used) which is network reachable and healthy.

StorageGroupPolicy = <Storage Group Policy Name> Storage Group Policy determines how Storage resources (from the 'Storage' directive) are being chosen from the Storage list. If no StoragePolicy is specified Bacula always tries to use first available Storage from the provided list. If the first few storage daemons are unavailable due to network problems; broken or unreachable for some other reason, Bacula will take the first one from the list (sorted according to the policy used) which is network reachable and healthy. Currently supported policies are:

ListedOrder - This is the default policy, which uses first available storage from the list provided.

LeastUsed - This policy scans all storage daemons from the list and chooses the one with the least number of jobs being currently run.

FreeSpace - This policy queries each Storage Daemon in the list for its FreeSpace (as a sum of devices specified in the SD config) and sorts the list according to the FreeSpace returned, so that first item in the list is the SD with the biggest amount of FreeSpace while the last one in the list is the one with the least amount of FreeSpace available.

LastBackedUpTo - This policy ensures that job is backed up to the storage where the same job (of the same level, i.e. Full or Incremental) has been backed up the longest time ago. The goal is to split the jobs to improve redundancy.

FreeSpaceLeastUsed - This policy ensures that job is backed up to the storage with more free space and less running jobs than others. Within the candidate storages, the least used will be selected. The candidate storages are determined by the **StorageGroupPolicyThreshold** directive. The storage will be a candidate if its free space is comprised between MaxFreeSpace-StorageGroupPolicyThreshold and MaxFreeSpace, where MaxFreeSpace is the highest value of free space for all storages in the group.

Example:

```
with StorageGroupPolicyThreshold=100MB

and storages free space being:
Storage1 = 500GB free
Storage2 = 200GB free
Storage3 = 400GB free
storage4 = 500GB free

In this case MaxFreeSpace=500GB.

Storage 1, 4 and 3 are candidates.

If 5 jobs are running on Storage1, 2 on Storage4, and 3 on Storage3, then Storage4
will be the selected storage.
```

StorageGroupPolicyThreshold = <Threshold size> Used in conjunction with the FreeSpaceLeastUsed StorageGroupePolicy to specify the range of free space Candidate storages.

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

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

By default, the watchdog thread will kill any Job that has run more than 200 days. The maximum watchdog timeout is independent of MaxRunTime and cannot be changed.

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

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

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

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

Maximum Spawned Jobs = <nb> The Job resource now permits specifying a number of **Maximum Spawn Jobs**. The default is **600**. This directive can be useful if you have big hardware and you do a lot of Migration/Copy jobs which start at the same time.

Maximum Bandwidth = <speed> The speed parameter specifies the maximum allowed bandwidth in **bytes** per second that a job may use. You may specify the following (case-insensitive) speed parameter modifiers: **kb/s** (1,000 bytes per second), **k/s** (1,024 bytes per second), **mb/s** (1,000,000 bytes per second), or **m/s** (1,048,576 bytes per second).

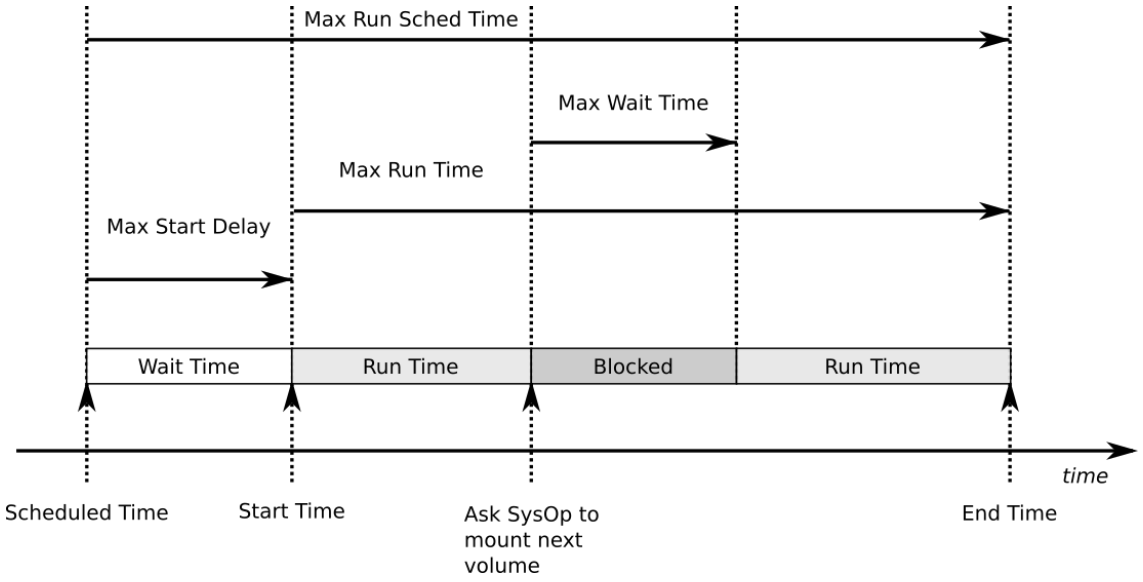


Fig. 1: Job time control directives

The use of TLS, TLS PSK, CommLine compression and Deduplication can interfere with the value set with the Directive.

This functionality affects only the data transfers between File Daemon and Storage Daemon, and was introduced with Bacula 6.0.0.

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

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

Note: VirtualFull job is not a real backup job. A VirtualFull will merge exiting jobs to create a new virtual Full job in the catalog and will copy the exiting data to new volumes.

The Client is not used in a VirtualFull job, so when using this directive, the Job that was supposed to run and save recently modified data on the Client will not run. Only the next regular Job defined in the Schedule will backup the data. It will not be possible to restore the data that was modified on the Client between the last Incremental/Differential and the VirtualFull.

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

If the directive is set to **no**, the Storage daemon will prefer finding an unused drive, otherwise, each job started will

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

Despite the above, we recommend against setting this directive to *no* since it tends to add a lot of swapping of Volumes between the different drives and can easily lead to deadlock situations in the Storage daemon.

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

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

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

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

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

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

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

You can use following console command: **delete, disable, enable, estimate, list, llist, memory, prune, purge, reload, status, setdebug, show, time, trace, update, version, .client, .jobs, .pool, .storage**.

See the console chapter for more information. You need to specify needed information on command line, nothing will be prompted.

Example:

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

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

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

Table 1: Table 12.1: Options for Run Script

Option	Value	Default	Information
Runs On Success	Yes / No	Yes	Run command if JobStatus is successful
Runs On Failure	Yes / No	No	Run command if JobStatus isn't successful
Runs On Client	Yes / No	Yes	Run command on client
Runs When	Before After Always Never AfterVSS AtJobCompletion Queued	Never	When to run commands
Fail Jon On Error	Yes / No	Yes	Fail job if script returns something different from 0
Command	String		Path to your script
Console	String		Console command
Timeout	Number		Timeout for the command (default 0)

Important: Regarding the Runs on Client option, scripts will run on Client only with Jobs that use a Client (Backup, Restore, some Verify Jobs). For other Jobs (Copy, Migration, Admin), RunsOnClient should be set to No.

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

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

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

```
%% = %
%b = Job Bytes
%c = Client's name
%C = If the job is a Cloned job (Only on director side)
%d = Daemon's name (Such as host-dir or host-fd)
%D = Director's name (Also valid on file daemon)
%e = Job Exit Status
%E = Non-fatal Job Errors
%f = Job FileSet (Only on director side)
%F = Job Files
%h = Client address
%i = JobId
%I = Migration/Copy JobId (Only in Copy/Migrate Jobs)
%j = Unique Job id
%l = Job Level
%n = Job name
%o = Job Priority
%p = Pool name (Only on director side)
%P = Current PID process
%R = Read Bytes
%s = Since time
```

(continues on next page)

(continued from previous page)

```
%S = Previous Job name (Only on file daemon side)
%t = Job type (Backup, ...)
%v = Volume name (Only on director side)
%w = Storage name (Only on director side)
%x = Spooling enabled? ("yes" or "no")
```

Some character substitutions are not available in all situations. The Job Exit Status code %e edits the following values:

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

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

You can use these following shortcuts:

Table 2: Table 12.2: RunScript shortcuts

Keyword	Runs On Succes	Runs On Failure	FailJob On Error	Runs On Client	Runs When
Run Before Job			Yes	No	Before
Run After Job	Yes	No		No	After
Run After Filed Job	No	Yes		No	After
Client Run Before Job				Yes	Before
Cient Run After Job	Yes	No	No	Yes	After

Examples:

```
RunScript {
    RunsWhen = Before
    FailJobOnError = No
    Command = "systemctl stop postgresql"
}
RunScript {
    RunsWhen = After
    RunsOnFailure = yes
    Command = "systemctl start postgresql"
}
```

Notes about the Run Queue Advanced Control with RunsWhen=Queued

It is possible to have an advanced control of the Bacula Director run queue. When a new starting Job is added to the run queue, the Director will control a certain number of conditions to let the Job start. In the condition list, we can find :

- Execution time (**when** parameter)
- Maximum Concurrent Jobs (for Director, Storage, Client, etc.)

- Priority
- etc.

By default, when all the conditions are met, the Director will move the Job to the active run queue and start the Job. The maximum execution time of the script is 15s by default.

It is possible to add custom conditions via a RunScript defined at the **RunsWhen** level **Queued** via the exit code of the script. The Director will determine if a Job must stay in the queue, or can analyze the other conditions. The following exit code can be used by the script:

- **0** Job can run, the script can be executed again if other conditions are not met.
- **1** Job must wait, the script will be executed again in 90 seconds.
- **2** Job will be canceled.
- **-1** Job can run, no more call of the script will be done while trying to acquire resources.

It can be used to control very precisely the Job execution flow. The output of the script is sent to the Job log.

For example, you might want to block new jobs to start during a maintenance. The maintenance mode is represented by a file on disk.

```
#!/bin/sh

if [ -f /opt/bacula/working/maintenance-mode ]; then
   echo "System under maintenance..."
   exit 1
fi
exit 0
```

```
 Job {
  Name = Backup1
  RunScript {
    RunsWhen = Queued
    RunsOnClient = no
    FailJobOnError = no
    Command = /opt/bacula/bin/maintenance-check.sh
  }
  JobDefs = Defaults
  Client = xxxx-fd
  FileSet = FS_xxxx
}
```

In the following example, the script will control the number of Jobs running for a given Client, but not take in account the restore Jobs.

```
#!/usr/bin/perl -w
use strict;
use JSON;
my $client = shift || '';
my $status = `echo -ne ".api 2 api_opts=j\n.status dir running client=
↪$client\n" | bconsole -u10 | grep '{'`;
my $info = JSON::decode_json($status);
my $nb_running = scalar( grep { $_->{status} eq 'R' && $_->{type} eq 'B
↪' }  @{ $info->{running}} );
```

(continues on next page)

(continued from previous page)

```
if ($nb_running >= 10) {
   print("Found $nb_running Jobs for $client\n");
   exit 1;
}
exit 0;
```

```
Job {
 Name = Backup1
 RunScript {
   RunsWhen = Queued
   RunsOnClient = no
   FailJobOnError = no
   Command = "/opt/bacula/bin/running-jobs.sh %c"
 }
 JobDefs = Defaults
 Client = xxxx-fd
 FileSet = FS_xxxx
 }
}
```

Notes about ClientRunBeforeJob

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

Special Shell Considerations

A "Command =" can be one of:

- The full path to an executable program.
- The name of an executable program that can be found in the $PATH
- A *complex* shell command in the form of: "sh -c \"your commands go here\""

Special Windows Considerations

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

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

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

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

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

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


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

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

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

The special characters

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

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

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

Suggestions for running on Windows machines with the native Windows File Daemon:

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

   ```
   ClientRunBeforeJob = "c:/bacula/bin/systemstate.bat"
   ```

   rather than DOS/Windows form:

   ```
   ClientRunBeforeJob = "c:/.bat" # INCORRECT
   ```

For Windows, note that there are certain limitations:

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

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

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

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

1. If all of the following conditions are met, then quote characters on the command line are preserved:
   - no /S switch
   - exactly two quote characters
   - no special characters between the two quote characters, where special is one of:

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

* there are one or more whitespace characters between the the two␣
↪quote
  characters
* the string between the two quote characters is the name of an␣
↪executable file.
```

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

The following example of the use of the Client Run Before Job directive was submitted by a user:

You could write a shell script to back up a DB2 database to a FIFO. The shell script is:

```
#!/bin/sh
# ===== backupdb.sh
DIR=/u01/mercuryd
mkfifo $DIR/dbpipe
db2 BACKUP DATABASE mercuryd TO $DIR/dbpipe WITHOUT PROMPTING &
sleep 1
```

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

```
Client Run Before Job = "su - mercuryd -c /u01/mercuryd/backupdb.sh '
```

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

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

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

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

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

it's equivalent to :

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


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

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

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

Note:

If you wish that your script will run regardless of the exit status of the Job, you can use this:

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

An example of the use of this directive is given in the Tips chapter of the Bacula Enterprise Problems Resolution Guide.

Client Run Before Job = <Command> This directive is the same as **Run Before Job** except that the program is run on the client machine. The same restrictions apply to Unix systems as noted above for the **RunScript**. **ClientRunBeforeJob** can be used with **Backup** and **Restore** jobs.

Client Run After Job = <Command> The specified is run on the client machine as soon as data spooling is complete in order to allow restarting applications on the client as soon as possible. **ClientRunBeforeJob** can be used with Backup and Restore jobs.

Note: See the notes above in **RunScript** concerning Windows clients.

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

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

Spool Data = <yes|no> If this directive is set to **yes** (default **no**), the Storage daemon will be requested to spool the data for this Job to disk rather than write it directly to the Volume (normally a tape).

Thus the data is written in large blocks to the Volume rather than small blocks. This directive is particularly useful when running multiple simultaneous backups to tape. Once all the data arrives or the spool files' maximum sizes are reached, the data will be despooled and written to tape.

Spooling data prevents interleaving date from several job and reduces or eliminates tape drive stop and start commonly known as "shoe-shine".

We don't recommend using this option if you are writing to a disk file using this option will probably just slow down the backup jobs.

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

Spool Attributes = <yes|no> The default is set to **yes**, 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. If set to **no** the File attributes are sent by the Storage daemon to the Director as they are stored on tape.

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

SpoolSize=bytes where the bytes specify the maximum spool size for this job. The default is take from Device Maximum Spool Size limit.

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

Add Prefix = <directory> This directive applies only to a Restore job and specifies a prefix to the directory name of all files being restored.

Add Suffix = <extention> This directive applies only to a Restore job and specifies a suffix to all files being restored.

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

Strip Prefix = <directory> This directive applies only to a Restore job and specifies a prefix to remove from the directory name of all files being restored.

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

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

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

RegexWhere = <expressions> This directive applies only to a Restore job and specifies a regex filename manipulation of all files being restored.

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

- **always** when the file to be restored already exists, it is deleted and then replaced by the copy that was backed up. This is the default value.
- **ifnewer** if the backed up file (on tape) is newer than the existing file, the existing file is deleted and replaced by the back up.
- **ifolder** if the backed up file (on tape) is older than the existing file, the existing file is deleted and replaced by the back up.
- **never** if the backed up file already exists, **Bacula** skips restoring this file.

RestoreClient = <client-resource-name> The RestoreClient directive specifies the default Client (File Daemon) that will be used with the restore job. If this directive is not set then a default restore client will be set to a backup client as usual. It is possible to define a dedicated restore job and run an automatic (scheduled) restore tests of your backups which will be redirected to the restore test Client.

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

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

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

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

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

Reschedule Interval = <time-specification> If you have specified **Reschedule On Error = yes** and the job terminates in error, it will be rescheduled after the interval of time specified by **<time-specification>**. See the time specification formats in the Configure chapter for details of time specifications. If no interval is specified, the job will not be rescheduled on error. The default Reschedule Interval is **30 minutes** (**1800 seconds**).

Reschedule Times = <count> This directive specifies the maximum number of times to reschedule the job. If it is set to **zero** (**0**, the default) the job will be rescheduled an indefinite number of times.

Allow Incomplete Jobs = <yes|no> If this directive is disabled, and the job terminates in incomplete status, the data of the job will be discarded and the job will be marked in error. Bacula will treat this job like a regular job in error. The default is **yes**.

Allow Duplicate Jobs = <yes|no> A duplicate job in the sense we use it here means a second or subsequent job with the same name starts. This happens most frequently when the first job runs longer than expected because no tapes are available. The default is **yes**.

If this directive is enabled duplicate jobs will be run. If the directive is set to **no** then only one job of a given name may run at one time, and the action that **Bacula** takes to ensure only one job runs is determined by the other directives (see below).

If **Allow Duplicate Jobs** is set to **no** and two jobs are present and none of the three directives given below permit cancelling a job, then the current job (the second one started) will be cancelled.

Cancel Lower Level Duplicates = <yes|no> If **Allow Duplicate Jobs** is set to **no** and this directive is set to **yes**, **Bacula** will choose between duplicated jobs the one with the highest level. For example, it will cancel a previous Incremental to run a Full backup. It works only for Backup jobs. The default is **no**. If the levels of the duplicated jobs are the same, nothing is done and the other Cancel XXX Duplicate directives will be examined.

Cancel Queued Duplicates = <yes|no> If **Allow Duplicate Jobs** is set to **no** and if this directive is set to **yes** any job that is already queued to run but not yet running will be canceled. The default is **no**.

Cancel Running Duplicates = <yes|no> If **Allow Duplicate Jobs** is set to **no** and if this directive is set to **yes** any job that is already running will be canceled. The default is **no**.

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

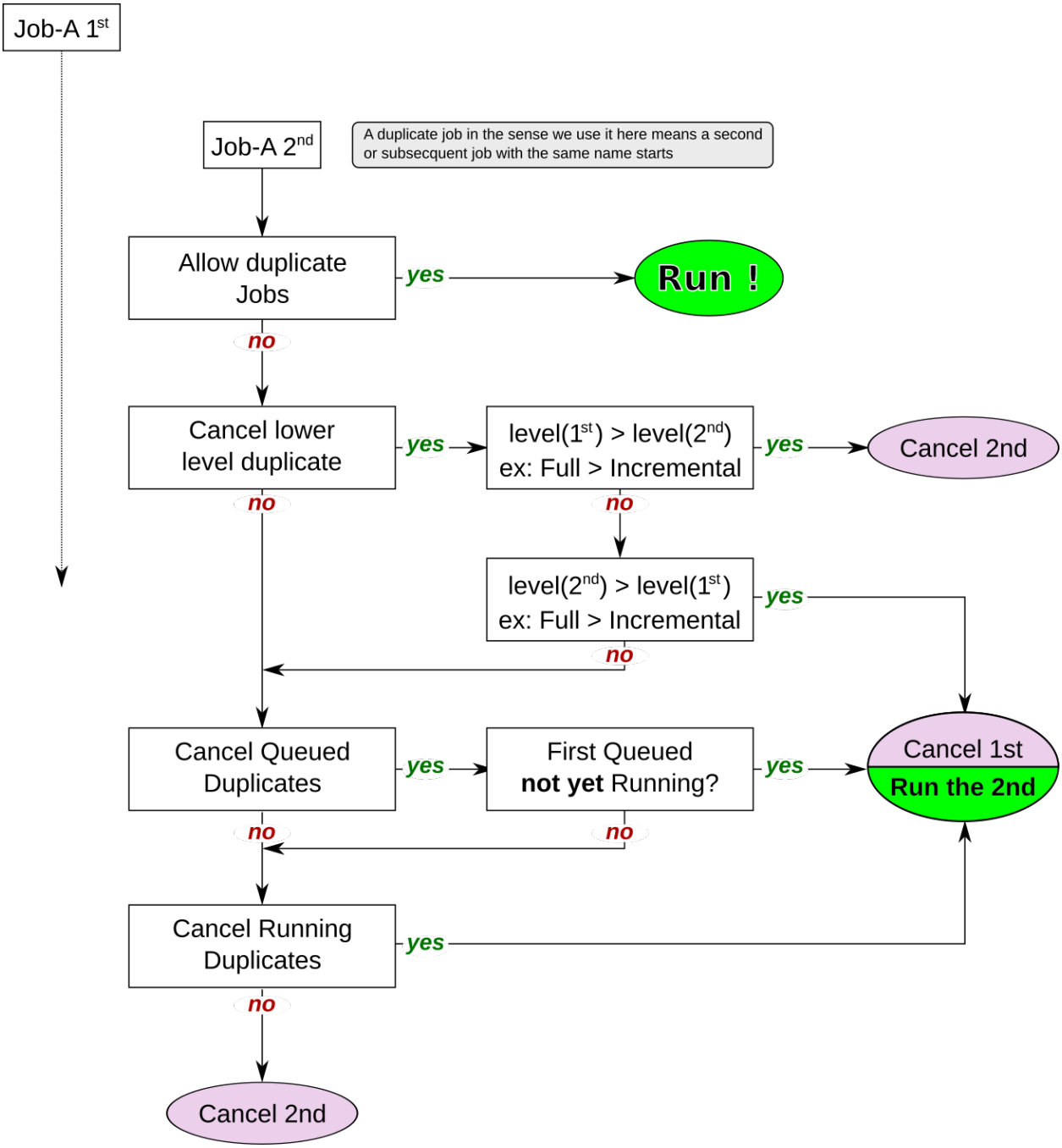


Fig. 2: Allow Duplicate Jobs usage

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

```
run = "LinuxHome level=%l since=\"%s\" storage=DiskAutochanger"
```

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

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

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

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

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

The default priority is **10**.

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

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

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

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

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

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

```
Job {
    Name = "LinuxHome"
    Type = Backup
    Level = Incremental   # default
    Client = bacula-fd
    FileSet="LinuxHome-fileset"
    Storage = DiskAutochanger
    Pool = DiskBackup365d
    Schedule = "Daily-schedule"
    Messages = Standard
}
```

CheckMalware = <yes|No> When set to **yes** (default **no**), the job will check the files recorded in the catalog against a Malware database. The Malware database (see MalwareDatabaseCommand) will be updated automatically when needed. To use the Malware detection, the FileSet must be configured with **Signature=MD5** or **Signature=SHA256**. See the MalwareDetection section of this manual for more information.

See also:

Go back to:

- *Director Resource*

Go to:

- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

JobDefs Resource

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

See also:

Go back to:

- *Director Resource*
- *Job Resource*

Go to:

- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Schedule Resource

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

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

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

Enabled = <yes|no> This directive allows you to enable or disable the resource.

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

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

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

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

- **Level=Full** is all files in the FileSet whether or not they have changed.
- **Level=Incremental** is all files that have changed since the last backup.
- **Pool=Weekly** specifies to use the Pool named **Weekly**.
- **Storage=DiskAutochanger** specifies to use **DiskAutochanger** for the storage device.
- **Messages=Verbose** specifies to use the **Verbose** message resource for the Job.
- **FullPool=Full** specifies to use the Pool named **Full** if the job is a full backup, or is upgraded from another type to a Full backup.
- **DifferentialPool=Differential** specifies to use the Pool named **Differential** if the job is a differential backup.
- **IncrementalPool=Incremental** specifies to use the Pool named **Incremental** if the job is an incremental backup.
- **Next Pool = <pool-specification>** The **Next Pool** directive specifies the pool to which Jobs will be migrated.
- **Priority = <number>** This directive permits you to control the order in which your jobs will be run by specifying a positive non-zero number. The higher the number, the lower the job priority. Assuming you are not running concurrent jobs, all queued jobs of priority 1 will run before queued jobs of priority 2 and so on, regardless of the original scheduling order.

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

  The default priority is **10**.

  See Priority for more information.

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

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

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

month and all hours.

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

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

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

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

According to the NIST, 12am and 12pm are ambiguous and can be defined to anything. However, 12:01am is the same as 00:01 and 12:01pm is the same as 12:01, so **Bacula** defines 12am as 00:00 (midnight) and 12pm as 12:00 (noon). You can avoid this abiguity (confusion) by using 24 hour time specifications (i.e. no am/pm).

An example schedule resource that is named **WeeklyCycle** and runs a job with level full each Sunday

at 2:05am and an incremental job Monday through Saturday at 2:05am is:

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

An example of a possible monthly cycle is as follows:

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

The first of every month:

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

The last day of February (28th or 29th), on May 31st and September 30rd at 20:00. In **show schedule** output, `LastDay is printed as "31"`

```
Schedule Name = "Last"
Run = Level=Full on lastday Feb, May, Sep at 20:00
```

Every 10 minutes:

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

Technical Notes on Schedules

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

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

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*

Go to:

- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

FileSet Resource

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

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

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

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

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

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

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

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

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

If this directive is set to **yes**, any changes you make to the FileSet Include or Exclude lists, will not force a Full during subsequent backups. Note that any changes to Options resources in the FileSet are not considered by this directive. You can use the Accurate mode for this to be treated correctly, or schedule a new Full backup manually.

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

Enable VSS = <yes|no> If this directive is set to **yes** the File daemon will be notified that the user wants to use a VSS backup for this Job. The default is **yes**.

This directive is effective only for `VSS` enabled Win32 File daemons. It permits a consistent copy of open files to be made for cooperating writer applications, and for applications that are not `VSS` aware, **Bacula** can at least access open files.

As of Bacula version 10.2, only the VSS snapshot will be used to identify files to back up. In prior versions, files were searched in the "regular" file system and backed up from the snapshot.

In earlier versions of Bacula, a VSS snapshot was only created on Windows drives that were explicitly mentioned in a **File List**; in many cases, this is no longer necessary.

Enable Snapshot = <yes|no> If this directive is set to **yes** the File Daemon will be notified that the user wants to use the Snapshot Engine for this job. The default is **no**. This directive is effective only for Snapshot enabled Unix File Daemons. It permits a consistent copy of open files to be made for cooperating applications. The `bsnapshot` tool should be installed on the Client.

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

Exclude {<file-list>}

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

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

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

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

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

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


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

on a Unix system where **/usr** is a subdirectory (rather than a mounted filesystem) will cause **/usr** to be backed up twice.

Please take note of the following items in the FileSet syntax:

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

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

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

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

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

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

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

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

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

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

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

compression at the same time you have hardware compression turned on, your files may actually take more space on the volume.

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

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

You can overwrite this option per Storage resource with AllowCompression option.

- **compression=LZO** All files saved will be software compressed using the `LZO` compression format. The compression is done on a file by file basis by the File Daemon. Everything else about `GZIP` is true for LZO.

  `LZO` provides much faster compression and decompression speed but lower compression ratio than `GZIP`. If your CPU is fast enough you should be able to compress your data without making the backup duration longer.

  Note: Bacula only use one compression level `LZO1X-1` specified by LZO.

  You can overwrite this option per Storage resource with AllowCompression option.

  ---

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

signature=SHA256 A SHA256 signature will be computed for all files saved. Adding this option generates about 5% extra overhead for each file saved. In addition to the additional CPU time, the SHA256 signature adds 44 more bytes per file to your catalog.

signature=SHA512 A SHA256 signature will be computed for all files saved. Adding this option generates about 5% extra overhead for each file saved. In addition to the additional CPU time, the SHA512 signature adds 87 more bytes per file to your catalog.

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

dedup=none||storage|bothsides The **Dedup** Fileset option can use the following values:

- **storage** All the deduplication work is done on Storage Daemon side if the device type is **dedup** (default value).
- **none** Force File Daemon and Storage Daemon to not use deduplication even if the device type is **dedup**.
- **bothsides** The deduplication work is done on the FD and the SD if the device type is **dedup**.

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

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

- **A** Always backup files

- **M** Look mtime/ctime like normal incremental backup
- **o** Check signature if other options would cause a backup, and if the signature is unchanged, back up only file metadata.

The 'o' option should be used in conjunction with one of the signature checking options (1, 2, 3, or 5). When the option is specified, the signature is computed only for files that have one of the other accurate options specified triggering a backup of the file (for example an inode change, a permission change, etc...). In cases where only the file's metadata has changed (ie: the signature is identical), only the file's attributes will be backed up. If the file's data has been changed (hence a different signature), the file will be backed up in the usual way.

This can save considerable space when backing up files which have only metadata part changed (e.g. changed permissions). The 'pinug2o' options could be used, which will then cause the File Daemon to compare permision bits, inotes, number of links, owner user and group and if any of it changes it will also compute the file's singature to verify if only metadata need to be backed up or is it needed to back up the file's contents.

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

- **i** compare the inodes
- **p** compare the permission bits
- **n** compare the number of links
- **u** compare the user id
- **g** compare the group id
- **s** compare the size
- **a** compare the access time
- **m** compare the modification time (st_mtime)
- **c** compare the change time (st_ctime)
- **d** report file size decreases
- **5** compare the MD5 signature
- **1** compare the SHA1 signature
- **2** compare the SHA256 signature
- **3** compare the SHA512 signature
- **o** Save only the metadata of a file when possible (available since Bacula Enterprise 12.8.2)

The 'o' option should be used in conjunction with one of the signature checking options (1, 2, 3, or 5). When the option is specified, the signature is computed only for files that have one of the other accurate options specified triggering a backup of the file (for example an inode change, a permission change, etc...). In cases where only the file's metadata has changed (ie: the signature is identical), only the file's attributes will be backed up. If the file's data has been changed (hence a different singature), the file will be backed up in the usual way

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

To save some space when backing up files which have only metadata part changed (e.g. changed permissions) the 'pino5' options could be used (or some other combination with the 'o' flag). It will then compare permision bits, inotes, number of links and if any of it changes it will also compute file's singature to verify if only metadata need to be backed up or is it needed to update file's contents.

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

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

Note: In older versions of **Bacula**, the above message was of the form:

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

If you wish to backup multiple filesystems, you can explicitly list each filesystem you want saved. Otherwise, if you set the onefs option to **no**, **Bacula** will backup all mounted file systems (i.e. traverse mount points) that are found within the **FileSet**. Thus if you have NFS or Samba file systems mounted on a directory listed in your FileSet, they will also be backed up. Normally, it is preferable to set **onefs=yes** and to explicitly name each filesystem you want backed up. Explicitly naming the filesystems you want backed up avoids the possibility of getting into a infinite loop recursing filesystems. Another possibility is to use **onefs=no** and to set **fstype=ext4, ...**. See the example below for more details.

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

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

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

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

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


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

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

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

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

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

Note: The `portable` directive shouldn't be required in recent Windows/Linux versions as recent Linux versions can handle files with win32 format.

---

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

sparse=<yes|no> Enable special code that checks for sparse files such as created by ndbm. The default is **no**, so no checks are made for sparse files. You may specify **sparse=yes** even on files that are not sparse file. No harm will be done, but there will be a small additional overhead to check for buffers of all zero, and if there is a 32K block of all zeros (see below), that block will become a hole in the file, which may not be desirable if the original file was not a sparse file.

- **Restrictions: Bacula** reads files in 64K buffers. If the whole buffer is zero, it will be treated as a sparse block and not written to the target Volume. However, if any part of the buffer is non-zero, the whole buffer will be written, possibly including large ranges of zeroes. As a consequence, **Bacula**'s detection of sparse blocks is in 64K increments rather than the system block size. If anyone considers this to be a real problem, please send in a request for change with the reason.

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

may be important. If you suspect you have sparse files, you should benchmark the difference or set sparse for only those files that are really sparse.

You probably should not use this option on files or raw disk devices that are not really sparse files (i.e. have holes in them).

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

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

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

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

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

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

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

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

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

---

checkfilechanges=<yes|no> If enabled, the Client will check size, age of each file after their backup to see if they have changed during backup. If time or size mismatch, an error will raise. The default value is **no**.

```
bacula-fd: Client1.2023-03-31_09.46.21 Error: /tmp/test mtime changed during␣
↪backup.
```

In general, it is recommended to use this option to backup regular files. It is not compatible with fifo or plugin streams.

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

a case, set **hardlinks=no** and hard links will not be backed up. Note, using this option will most likely backup more data and on a restore the file system will not be restored identically to the original.

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

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

enhancedwild=<yes|no> The `Enhanced Wild` directive controls how path name matching with wildcards is done.

The default is `no`, which makes wildcards not match match path separators. If set to `yes`, an asterisk, question mark, or a bracketed slash will also be matched by a slash. In other words, wildcards will then span path hierarchy.

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

It is recommended to enclose the string in double quotes.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the Utilities command for more information. You can also test your full FileSet definition by using the estimate command.

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

It is recommended to enclose the string in double quotes.

You may want to test your expressions prior to running your backup by using the bwild program. Please see the Utilities command for more information. You can also test your full FileSet definition by using the estimate command.

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

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program.

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

It is recommended to enclose the string in double quotes.

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

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

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

It is recommended to enclose the string in double quotes.

The regex libraries differ from one operating system to another, and in addition, regular expressions are complicated, so you may want to test your expressions prior to running your backup by using the bregex program.

exclude=<string> The default is **No**. When enabled, any files matched within the Options will be excluded from the backup.

aclsupport=<yes|no> The default is **no**. If this option is set to yes, and you have the needed `POSIX` ACL support installed, **Bacula** will backup the file and directory Unix ACLs as defined in IEEE Std 1003.1e draft 17 and "POSIX.1e" (abandoned). This feature is available on Unix and Linux systems only and requires the platform ACL library. **Bacula** is automatically compiled with ACL support if the **libacl** library is installed on your system (shown in config.out). While restoring the files **Bacula** will try to restore the ACLs, if there is no ACL support available on the system as a whole or the destination file system, **Bacula** restores the files and directories but not the ACL information.

Linux systems usually have ACL support available, if not installed by default. The file systems often need to be mounted with explicit options to enable ACLs.

For other Unix operating systems there is typically support for either POSIX ACLs or the more extensible NFSv4 ACLs.

The ACL stream format between Operation Systems is **not** compatible so for example an ACL saved on Linux cannot be restored on Solaris.

The following Operating Systems are currently supported:

1. AIX (pre-5.3 (POSIX) and post 5.3 (POSIX and NFSv4) ACLs)
2. Darwin
3. FreeBSD (POSIX and NFSv4/ZFS ACLs)
4. HPUX
5. IRIX
6. Linux (POSIX and GPFS)
7. Solaris (POSIX and NFSv4/ZFS ACLs)
8. Tru64

xattrsupport=<yes|no> The default is **no**. If this option is set to yes, and your operating system support either so called Extended Attributes or Extensible Attributes **Bacula** will backup the file and directory XATTR data. This feature is available on Unix and Linux only and depends on support of some specific library calls in libc.

The XATTR stream format between Operating Systems is **not** compatible so an XATTR saved on Linux cannot for example be restored on Solaris.

On some operating systems ACLs are also stored as Extended Attributes (Linux, Darwin, FreeBSD) **Bacula** checks if you have the aclsupport option enabled and if so will not save the same info when saving extended attribute information. Thus ACLs are only saved once.

The following Operating Systems are currently supported:

1. AIX (Extended Attributes)

2. Darwin (Extended Attributes)
3. FreeBSD (Extended Attributes)
4. IRIX (Extended Attributes)
5. Linux (Extended Attributes)
6. NetBSD (Extended Attributes)
7. Solaris (Extended Attributes and Extensible Attributes)
8. Tru64 (Extended Attributes)

On Linux systems, extended attribute support depends on file system as well as mount options. Bacula Enterprise version 10.2 introduced specific support for CIFS file systems (Windows shares): If Linux kernel and CIFS file system provide the necessary functionality, the special `system.cifs_acl` extended attribute can be backed up. Restoring those extended attributes / ACLs will not recreate an identical file tree, as the ownership of the restored files will be determined by the CIFS mount option. ACLs referring to determined users will be working correctly. As such, we recommend relying on this feature only when data restores and ACLs are the goal and ownership information can reasonably well be fixed in a manual step. Please consult mount.cifs(8) for technical details.

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

fstype=filesystem-type-list This option ensures that the FD, while processing files with this option block effective, will only descend into filesystems of types mentioned in the `Filesystem-type-list`. This is most useful in combination with `One FS = no` set.

This directive may appear multiple times, and all list elements will be added, i.e. a specification of:

```
FS Type = ext4, xfs
FS Type = ntfs
```

will result in all three mentioned file system types being accepted.

On Windows, this functionality is not available, but see Drive Type.

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

The permitted drivetype names are:

removable, fixed, remote, cdrom, ramdisk

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

This option is not implemented in Unix/Linux systems.

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

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

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

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

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

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

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

  This allows you to have a job that, for example, includes all the local partitions even if you change the partitioning by adding a disk. The examples below show you how to do this. However, note two things:

  1. if you want the local filesystems, you probably should be using the **fstype** directive, which was added in version 1.36.3 and set **onefs=no**.
  2. the exact syntax of the command needed in the examples below is very system dependent. For example, on recent Linux systems, you may need to add the -P option, on FreeBSD systems, the options will be different as well.

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

  As an example:

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

  will produce a list of all the local partitions on an old Red Hat Linux system.

Note: The above line was split, but should normally be written on one line. Quoting is a real problem because you must quote for **Bacula** which consists of preceding every \ and every " with a \, and you must also quote for the shell command.

  In the end, it is probably easier just to execute a small file with:

```
Include {
    Options {
```

(continues on next page)

(continued from previous page)

```
        signature=MD5
        }
    File = "|my_partitions"
    }
```

where my_partitions has:

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

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

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

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

If you know what filesystems you have mounted on your system, e.g. for Red Hat Linux normally ext4 and ext3, you can backup all local filesystems using something like:

```
Include {
    Options {signature = SHA1; onfs=no; fstype=ext4 }
    File = /
    }
```

On Windows, the command is executed with "cmd /c" prefix, and it is recommended to keep the path of the scripts as simple as possible (without spaces for example). The exact quoting and escaping sequence of the string can be difficult to determine.

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

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

If you precede the less-than sign (<) with a backslash as in \<, the file-list will be read on the Client machine instead of on the Director's machine.

Note: If the filename is given within quotes, you will need to use two backslashes.

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

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

```
Include {
    Options {signature=MD5; sparse=yes }
    File = /dev/hdb6
}
```

  will backup the data in device **/dev/hdb6**. Note, the **/dev/hdb6** must be the raw partition itself. **Bacula** will not back it up as a raw device if you specify a symbolic link to a raw device such as may be created by the LVM Snapshot utilities.

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

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

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

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

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

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

```
# List of files to be backed up
FileSet {
    Name = "MyFileSet"
    Include {
        Options {
            signature = MD5
```

(continues on next page)

(continued from previous page)

```
        }
    File = /home
    Exclude Dir Containing = .excludeme
    }
}
```

will allow users to indicate that they don't want to have certain directories backed up. For example, with the above FileSet, if the user or sysadmin creates a file named **.excludeme** in specific directories, such as

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

then **Bacula** will not backup the two directories named:

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

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

FileSet Examples

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

Note: The first Include pulls in the contents of the file when **Bacula** is started (i.e. the @), and that file must have each filename to be backed up preceded by a **File =** and on a separate line.

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


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

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

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

```
FileSet {
    Name = "Full Set"
    Include {
        Options {
            Compression=GZIP
            signature=SHA1
            Sparse = yes
        }
    @/etc/backup.list
    }

    Include {
        Options {
            wildfile = "*.o"
            wildfile = "*.exe"
            Exclude = yes
        }
        File = /root/myfile
        File = /usr/lib/another_file
    }
    Exclude {             # don't add trailing /
        File = /tmp
    }
}
```

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

```
:math:`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 lmatou:/ 1554264 215884
1258056 15 lmatou:/home 2478140 1589952 760072 68 lmatou:/usr 1981000
1199960 678628 64 lpmatou:/ 995116 484112 459596 52 lpmatou:/home
19222656 2787880 15458228 16 lpmatou:/usr 2478140 2038764 311260 87
deuter:/ 4806936 97684 4465064 3 deuter:/home 4806904 280100 4282620
7 deuter:/files 44133352 27652876 14238608 67
```

And we see that there are a number of separate filesystems (**/ /boot /home /rescue/tmp** and **/tmp** not to mention

mounted systems). If you specify only in your Include list, **Bacula** will only save the Filesystem **/dev/hda5**. To save all filesystems except **/tmp** with out including any of the Samba or NFS mounted systems, and explicitly excluding a **/tmp**, **proc**, **.journal**, and **.autofsck**, which you will not want to be saved and restored, you can use the following:

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
            RegexFile = ".*"
        }
        File = /myfile
    }
}
```

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
            exclude = yes
        }
    File = /home
    }
}
```

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

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

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

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

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

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

Firstly, Bacula traverses the filesystem starting from the File = lines. It stops descending when a directory is excluded, so you must include all ancestor (higher level) directories of each directory containing files to be included.

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

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

```
FileSet {
    Name = "AllPictures"
    Include {
        File = "C:/Documents and Settings"
        Options {
            signature = SHA1
            verify = s1
            IgnoreCase = yes
            # Include all users' directories so we reach the inner ones. Unlike a
            # WildDir pattern ending in *, this RegExDir only matches the top-level
            # directories and not any inner ones.
            RegExDir = "^C:/Documents and Settings/[^/]+$"
            # Ditto all users' My Documents directories.
            WildDir = "C:/Documents and Settings/*/My Documents"
```

(continues on next page)

(continued from previous page)

```
            # Ditto all users' My Documents/My Pictures directories.
            WildDir = "C:/Documents and Settings/*/My Documents/My Pictures"
            # Include the contents of the My Documents/My Pictures directories and
            # any subdirectories.
            Wild = "C:/Documents and Settings/*/My Documents/My Pictures/*"
        }
        Options {
            Exclude = yes
            IgnoreCase = yes
            # Exclude everything else, in particular any files at the top level and
            # any other directories or files in the users' directories.
            Wild = "C:/Documents and Settings/*"
            }
        }
    }
```

In the above example, we setup a FileSet to backup a subset of a single directory (on Windows) based on filename.

```
FileSet {
    Name = "win-fileset"
    Include {
        Options {
            signature = MD5
            wildfile = "C:/File/backup_NDB_COUNT*"
        }
        # Exclude all files not matching the wildfile list above
    Options {
        Exclude = yes
        RegexFile = ".*"
        }
    File = C:/File
    }
}
```

Backing Up Raw Partitions

The following FileSet definition will backup a raw partition:

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

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

Excluding Files and Directories

You may also include full filenames or directory names in addition to using wild-cards and **Exclude=yes** in the Options resource as specified above by simply including the files to be excluded in an Exclude resource within the FileSet. It accepts wild-cards pattern, so for a directory, don't add a trailing /. For example:

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

The following fileset will not work as expected:

```
Fileset {
  Name = "Home-fileset"
  Description = "Home directory without File1* files"
  Include {
   Options {
    AclSupport = yes
    FsType = "xfs","ext2","ntfs","ext3","ext4","apfs","hfs"
    OneFs = no
    Signature = Sha1
    Sparse = yes
    XattrSupport = yes
   }
   Options {
     Exclude = yes
     WildDir = "/home/dir1/File1*"
  }
   File = "/"
  }
}
```

When parsing the Options blocks, Bacula will parse the first block and apply all the defined options if a file matches it. However, it will go through the second Options block (having Exclude = yes) and, even if the file doesn't match this block, the options from this second block are applied to the file that will be included in the backup. As there is no option defined, the default values are used which means "AclSupport = no, OneFS = yes, Signature = none, Sparse = no, xattrSupport = no" will be used. The options you wish to have applied for all files that will be included in the backup, should be in the last options block:


```
Fileset {
  Name = "Home-fileset"
  Description = "Home directory without File1* files"
  Include {
   Options {
     Exclude = yes
     WildDir = "/home/dir1/File1*"
  }
   Options {
    AclSupport = yes
    FsType = "xfs","ext2","ntfs","ext3","ext4","apfs","hfs"
    OneFs = no
    Signature = Sha1
    Sparse = yes
    XattrSupport = yes
   }
   File = "/"
  }
}
```

Windows FileSets

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

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

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

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

If you are having difficulties getting includes or excludes to work, you might want to try using the **estimate job=xxx listing** command.

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

A Windows Example FileSet

Note: For presentation purposes, the lines beginning with Data and Internet have been wrapped and should included on the previous line with one space.

```
This is my Windows 2022 fileset:

FileSet {
    Name = "Windows 2022"
    Include {
        Options {
            signature = MD5
            Exclude = yes
            IgnoreCase = yes
            # Exclude cache files
            WildDir = "[A-Z]:/ProgramData/Windows/Caches"
            WildDir = "[A-Z]:/Users/*/AppData"
            WildFile = "[A-Z]:/Users/*/ntuser.DAT"
            WildDir = "C:/ClusterStorage"
            # Some random bits of Windows we want to ignore
            WildDir = "[A-Z]:/Windows/system32/config"
            WildDir = "[A-Z]:/Windows/NTDS"
            WildDir = "[A-Z]:/Windows/cluster"
            # Temporary directories & files
            WildDir = "[A-Z]:/Windows/temp"
            WildDir = "[A-Z]:/Windows/Spool"
            WildFile = "*.tmp"
            # Recycle bins
            WildDir = "[A-Z]:/$RECYCLE.BIN"
            # Swap files
            WildFile = "[A-Z]:/pagefile.sys"
            # These are programs and are easier to reinstall than restore from
            # backup
            WildDir = "[A-Z]:/cygwin"
            WildDir = "[A-Z]:/Program Files/Grisoft"
            WildDir = "[A-Z]:/Program Files/Java"
            WildDir = "[A-Z]:/Program Files/Java Web Start"
            WildDir = "[A-Z]:/Program Files/JavaSoft"
            WildDir = "[A-Z]:/Program Files/Microsoft Office"
            WildDir = "[A-Z]:/Program Files/Mozilla Firefox"
            WildDir = "[A-Z]:/Program Files/Mozilla Thunderbird"
            WildDir = "[A-Z]:/Program Files/OpenOffice*"
            WildDir = "[A-Z]:/Program Files (x86)/Internet Explorer"
            WildDir = "[A-Z]:/Program Files (x86)/Windows Defender"
        }
```

(continues on next page)

(continued from previous page)

```
    File = "/"
    }
}
```

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

Testing Your FileSet

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

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

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

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

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

to give you a listing of all files that match. In the above example, it should be only files with names ending in .

Include All Windows Drives in FileSet

The `alldrives` Windows Plugin allows you to include all local drives with a simple directive. This plugin is available in the Windows 64 and 32 bit installer.

```
FileSet {
    Name = EverythingFS
    ...
    Include {
        Plugin = "alldrives"
    }
}
```

You exclude some specific drives with the `exclude` option.

```
FileSet Name = EverythingFS ... Include Plugin = "alldrives:
exclude=D,E"
```


The all-drives Windows plugin only considers regular drives (i.e. **C:/**, **D:/**, **E:/**). Mount points inside the directory tree will no be part of the snapshots unless you set **"OneFS = no"** in the block of the FileSet, which will cause **Bacula** to snapshot everything.

Microsoft VSS Writer Plugin

We provide a single plugin named **vss-fd.dll** that permits you to backup a number of different components on Windows machines. This plugin is available from **Bacula** Systems as an option.

Find more information about it here: vss-plugin

Contact the Support Team to find more information about our VSS products.

System State writers:

- Registry
- Event Logs
- COM+ REGDB (COM Registration Database)
- System (Systems files – most of what is under c:/windows and more)
- WMI
- NTDS (Active Directory)
- NTFRS (SYSVOL etc replication – Windows 2003 domains)
- DFS Replication (SYSVOLS etc replication – Windows 2008 domains)
- ASR Writer

This component is known to work.

- MSSQL databases
- Exchange

Each of the above specified Microsoft components can be backed up by specifying a different plugin option within the **Bacula** FileSet. All specifications must start with **vss:** and be followed with a keyword which indicates the writer, such as **/@SYSTEMSTATE/** (see below).

To activate each component, use the following:

- System State writers

```
Plugin = "vss:/@SYSTEMSTATE/"
```

  Note, exactly which subcomponents will be backed up depends on which ones you have enabled within Windows. For example, on a standard default Vista system only ASR Writer, COM+ REGDB, System State, and WMI are enabled.

- MSSQL databases (except those owned by Sharepoint if that plugin is specified)

```
Plugin = "vss:/@MSSQL/"
```

  The Microsoft literature says that the mssql writer is only good for snapshots and it needs to be enabled via a registry tweak or else the older MSDE writer will be invoked instead.

- Exchange (all exchange databases)


```
Plugin = "vss:/@EXCHANGE/"
```

The plugin directives must be specified exactly as shown above. A Job may have one or more of the **vss** plugins components specified.

Also ensure that the **vss-fd.dll** plugin is in the plugins directory on the FD doing the backup, and that the plugin directory config line is present in the FD's configuration file (**bacula-fd.conf**).

Backup

If everything is set up correctly as above then the backup should include the system state. The system state files backed up will appear in a **bconsole** or BAT(Bacula Administration Tool) restore like:

```
/@SYSTEMSTATE/
/@SYSTEMSTATE/ASR Writer/
/@SYSTEMSTATE/COM+ REGDBWriter/
```

etc.

Only a complete backup of the system state is supported at this time. That is it is not currently possible to just back up the Registry or Active Directory by itself. In almost all cases a complete backup is a good idea anyway as most of the components are interconnected in some way. Also, if an incremental or differential backup is specified on the backup Job then a full backup of the system state will still be done. The size varies according to your installation. The actual size depends on how many Windows components are enabled.

The system state component automatically respects all the excludes present in the FilesNotToBackup registry key, which includes things like **%TEMP%**, **pagefile.sys**, **hiberfil.sys**, etc. Each plugin may additionally specify files to exclude, eg the VSS Registry Writer will tell **Bacula** to not back up the registry hives under **C:WINDOWSsystem32config** because they are backed up as part of the system state.

Restore

In most cases a restore of the entire backed up system state is recommended. Individual writers can be selected for restore, but currently not individual components of those writers. To restore just the Registry, you would need to mark @SYSTEMSTATE (only the directory, not the subdirectories), and then do **mark Registry*** to mark the Registry writer and everything under it.

Restoring anything less than a single component may not produce the intended results and should only be done if a specific need arises and you know what you are doing, and not without testing on a non-critical system first.

To restore Active Directory, the system will need to be booted into Directory Services Restore Mode, an option at Windows boot time.

Only a non-authoritative restore of NTFRS/DFSR is supported at this time. There exists Windows literature to turn a DC restored in non-authoritative mode back into an authoritative Domain Controller. If only one DC exists it appears that Windows does an authoritative restore anyway.

Most VSS components will want to restore to files that are currently in use. A reboot will be required to complete the restore (eg to bring the restored registry online).

Starting another restore of VSS data after the restore of the registry without first rebooting will not produce the intended results as the "to be replaced next reboot" file list will only be updated in the "to be replaced" copy of the registry and so will not be actioned.

Example

Suppose you have the following backup FileSet:

```
@SYSTEMSTATE/
    System Writer/
        instance_{GUID}
        System Files/
    Registry Writer/
        instance_{GUID}
        Registry/
    COM+ REGDB Writer/
        instance_{GUID}
        COM+ REGDB/
    NTDS/
        instance_{GUID}
        ntds/
```

If only the Registry needs to be restored, then you could use the following commands in **bconsole:**

```
markdir @SYSTEMSTATE
cd @SYSTEMSTATE
markdir "Registry Writer"
cd "Registry Writer"
mark instance*
mark "Registry"
```

Windows Plugins Items to Note

- **Reboot Required after a Plugin Restore In general after any VSS plugin** is used to restore a component, you will need to reboot the system. This is required because in-use files cannot be replaced during restore time, so they are noted in the registry and replaced when the system reboots.
- **After a System State restore, a reboot will generally take longer** than normal because the pre-boot process must move the newly restored files into their final place prior to actually booting the OS.
- **One File from Each Drive needed by the Plugins must be backed up** At least one file from each drive that will be needed by the plugin must have a regular file that is marked for backup. This is to ensure that the main **Bacula** code does a snapshot of all the required drives. At a later time, we will find a way to accomplish this automatically.
- **Bacula does not Automatically Backup Mounted Drives** Any drive that is mounted in the normal file structure using a mount point or junction point will not be backed up by **Bacula**. If you want it backed up, you must explicitly mention it in a **Bacula file** directive in your FileSet.
- **When doing a backup that is to be used as a Bare Metal Recovery, do** not use the VSS plugin. The reason is that during a Bare Metal Recovery, VSS is not available nor are the writers from the various components that are needed to do the restore. You might do full backup to be used with a Bare Metal Recovery once a month or once a week, and all other days, do a backup using the VSS plugin, but under a different Job name. Then to restore your system, use the last Full non-VSS backup to restore your system, and after rebooting do a restore with the VSS plugin to get everything fully up to date.

This plugin is available as an option. Contact **Bacula** Systems to get access to the VSS Plugin packages and the documentation.

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*

Go to:

- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Client Resource

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

Client (or FileDaemon) Start of the Client directives.

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

Enabled = <yes|no> This directive allows you to enable or disable the resource. If the resource is disabled, the Client will not be used.

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

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

AllowFDConnections = <yes|no> When **AllowFDConnections** is set to **true**, the Director will accept incoming connections from the Client and will keep the socket open for a future use. The Director will no longer use the **Address** to contact the File Daemon. This configuration is useful if the Director cannot contact the FileDaemon directly. See the ConnectToDirector directive in the client configuration for more information. The default value is **no**.

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

Password = <password> This is the password to be used when establishing a connection with the File services, so the Client configuration file on the machine to be backed up must have the same password defined for this Director. This

directive is required. If you have either **/dev/random** or **bc** on your machine, **Bacula** will generate a random password during the configuration process, otherwise it will be left blank.

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

Snapshot Retention = <time-period-specification> The Snapshot Retention directive defines the length of time that **Bacula** will keep Snapshots in the Catalog database and on the Client after the Snapshot creation. When this time period expires, and if using the **snapshot prune** command, **Bacula** will prune (remove) Snapshot records that are older than the specified Snapshot Retention period and will contact the File Daemon to delete Snapshots from the system.

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

The default is **0 seconds**, Snapshots are deleted at the end of the backup. The Job **SnapshotRetention** directive overwrites the Client **SnapshotRetention** directive.

File Retention = <time-period-specification> The File Retention directive defines the length of time that **Bacula** will keep File records in the Catalog database after the End time of the Job corresponding to the File records. When this time period expires, and if **AutoPrune** is set to **yes**, **Bacula** will prune (remove) File records that are older than the specified File Retention period.

Note: This affects only records in the catalog database. It does not affect your archive backups.

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

The default is **60 days**.

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

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

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

The default is **180 days**.

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

Maximum Concurrent Jobs = <number> where **<number>** is the maximum number of Jobs with the current Client that can run concurrently. Note, this directive limits only Jobs for Clients with the same name as the resource in which it appears. Any other restrictions on the maximum concurrent jobs such as in the Director, Job, or Storage resources will also apply in addition to any limit specified here. The default is set to **1**, but you may set it to a larger number. If set to a large value, be careful to not have this value higher than the **Maximum Concurrent Jobs** configured in the resource in the Client/File daemon configuration file. Otherwise, backup jobs can fail due to the Director connection to FD be refused because MCJ was exceeded on FD side.

Maximum Bandwidth Per Job = <speed> The speed parameter specifies the maximum allowed bandwidth in bytes per second that a job may use when started for this Client. You may specify the following case-insensitive speed parameter modifiers: kb/s (1,000 bytes per second), k/s (1,024 bytes per second), mb/s (1,000,000 bytes per second), or m/s (1,048,576 bytes per second).

The use of TLS, TLS PSK, CommLine compression and Deduplication can interfere with the value set with the Directive.

This functionality only affects the data transfer from the File Daemon to the Storage Daemon.

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).

SD Calls Client = <yes|no> If the **SD Calls Client** directive is set to true in a resource any Backup, Restore, Verify Job where the client is involved, the client will wait for the Storage daemon to contact it. By default this directive is set to **false**, and the Client will call the Storage daemon as it always has. This directive can be useful if your Storage daemon is behind a firewall that permits outgoing connections but not incoming connections.

FD Storage Address = <address> Where **<address>** is a host name, a **FQDN**, or an **IP address**. The value specified here will be transmitted to the File Daemon instead of the address that the Director uses to contact the Storage Daemon. This FDStorageAddress will then be used by the File Daemon to contact the Storage Daemon. This is particularly useful if the File Daemon is in a different network domain than the Director or Storage Daemon. It is also useful in NAT or firewall environments.

```
Client {
     Name = client1
     Address = 65.1.1.5
     FD Port = 9102
 FD Storage Address = 10.0.0.1
     ...
     }
```

The normal way to handle this situation is to use a canonical name such as "storage-server" that will be resolved on the Director side as the `WAN` address and on the Client side as the `LAN` address. Name servers usually refer to this functionality as providing "views", i.e. zones with different contents depending on where a query originates.

It is possible to configure the **FDStorageAddress** in both the Storage or Client resource.

Note that using the Client **FDStorageAddress** directive will not allow to use multiple Storage Daemon, all Backup or Restore requests will be sent to the specified **FDStorageAddress**.

Note: TLS Directives in the Client resource of bacula-dir.conf

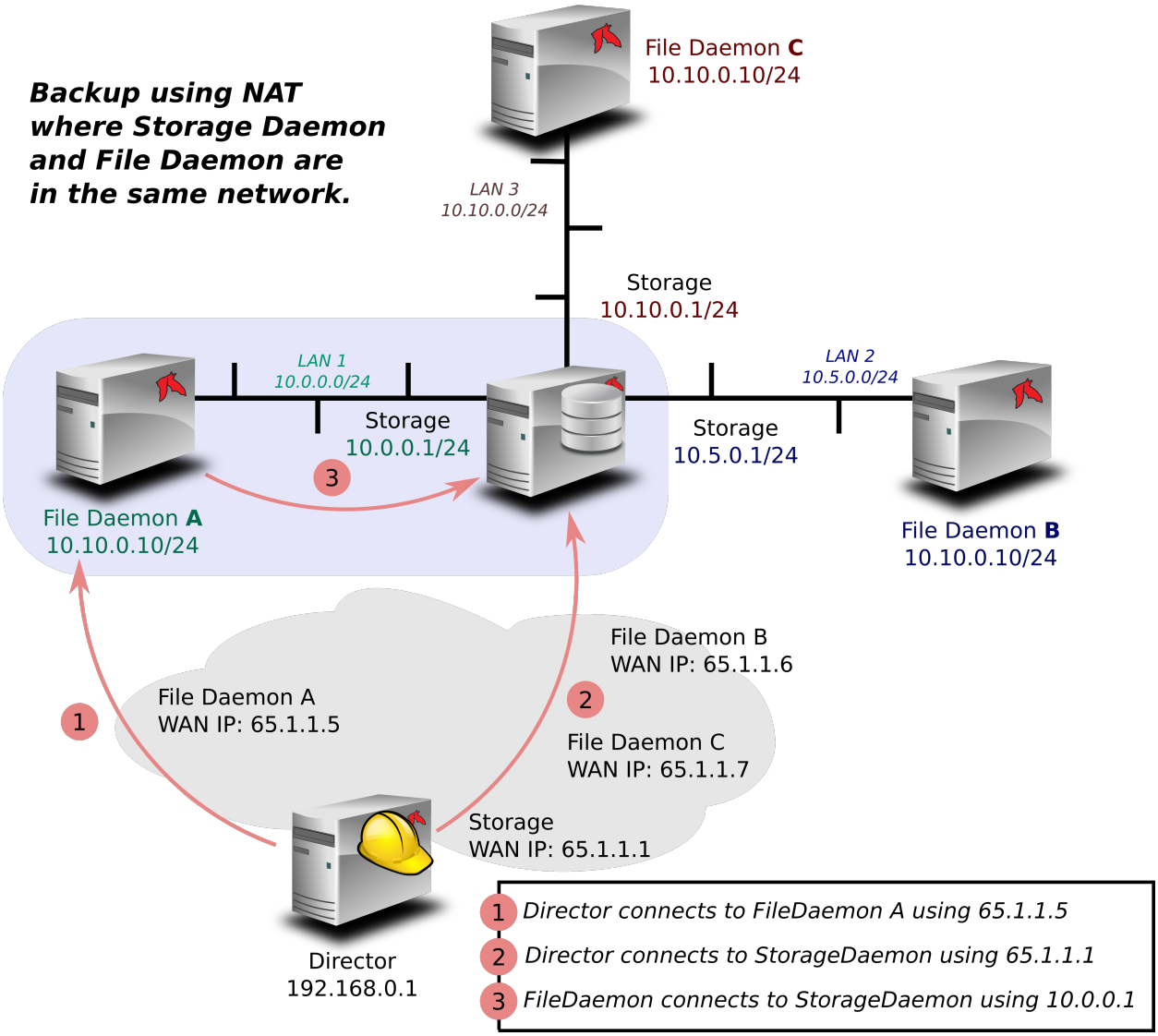
Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh**. The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

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

```
Client {
  Name = bacula-fd
  Address = bacula-client.example.com
  Catalog = BaculaCatalog
```

(continues on next page)



(continued from previous page)

```
  Password = strong_password
}
```

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*

Go to:

- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Storage Resource

The Storage resource defines which Storage devices, provided by Storage Daemons, are available for use by the Director.

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

The **Autochanger** resource type is nearly identical, for the differences and use cases see TechnicalReferenceDirAutochangerResource section.

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

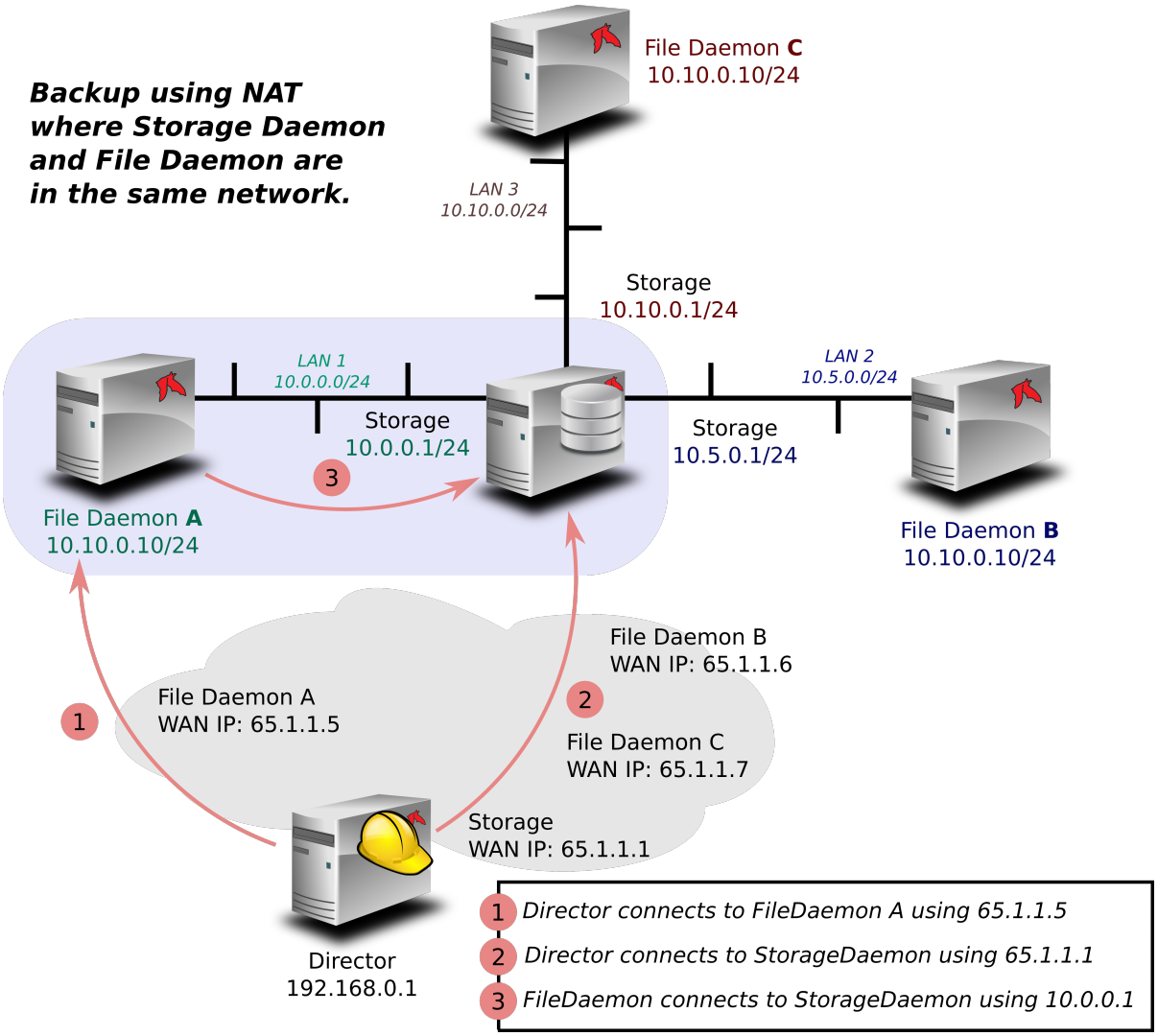
Enabled = <yes|no> This directive allows you to enable or disable a resource. When the resource is disabled, the storage device will not be used. To reuse it you must re-enable the resource.

Address = <address> Where the address is a host name, a **FQDN**, or an **IP Address**. Please note that the **<address>** as specified here will be transmitted to the File daemon who will then use it to contact the Storage Daemon. Hence, it is **not**, a good idea to use **localhost** as the name but rather a fully qualified machine name or an **IP Address**. This directive is required.

FD Storage Address = <address> Where the <address> is a host name, a **FQDN**, or an **IP Address**. The **<address>** specified here will be transmitted to the File Daemon instead of the address the Director uses to contact

the Storage Daemon. This FDStorageAddress will then be used by the File Daemon to contact the Storage Daemon. This directive particularly useful if the File daemon is in a different network domain than the Director or Storage daemon. It is also useful in **NAT** or firewall environments.

See the DirectorClientFdStorageAddress section for a full explanation.



SD Port = <port> Where port is the port to use to contact the storage daemon for information and to start jobs. This same port number must appear in the Storage resource of the Storage daemon's configuration file. The default is **9103**.

Password = <password> This is the password to be used when establishing a connection with the Storage services. This same password also must appear in the Director resource of the Storage daemon's configuration file. This directive is required. If you have either **/dev/random** or **bc** on your machine, **Bacula** will generate a random password during the configuration process, otherwise it will be left blank.

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

Device = <device-name> This directive specifies the Storage daemon's name of the device resource to be used for the storage. If you are using an Autochanger, the name specified here should be the name of the Storage daemon's Autochanger resource rather than the name of an individual device. This name is not the physical device name, but the

logical device name as defined on the **Name** directive contained in the or the 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**). Do not define two different Storage resource directives in the Director that point to the same Device in the Storage daemon. Doing so may cause the Storage daemon to block (or hang) attempting to open the same device that is already open. This directive is required.

Media Type = <MediaType> This directive specifies the Media Type to be used to store the data. This is an arbitrary string of characters up to 127 maximum that you define. It can be anything you want. However, it is best to make it descriptive of the storage media (e.g. File, LTO-7, LTO-8, ...). In addition, it is essential that you make the specification unique for each storage media type. If you have two **LTO** drives that have incompatible formats, or if you have a **LTO-7** drive and a **LTO-8** autochanger, you almost certainly should specify different **Media Types**. During a restore, assuming a **LTO-8** Media Type is associated with the Job, **Bacula** can decide to use any Storage daemon that supports Media Type **LTO-8** and on any drive that supports it.

If you are writing to disk Volumes, you must make doubly sure that each Device resource defined in the Storage daemon (and hence in the Director's conf file) has a unique media type. Otherwise, your restores may not work because **Bacula** will assume that you can mount any volume with the same Media Type any Device associated with that Media Type. This is possible with tape drives, but with disk drives, unless you are very clever you cannot mount a Volume in any directory – this can be done by creating an appropriate soft link.

Currently **Bacula** permits only a single Media Type per Storage and Device definition. Consequently, if you have a drive that supports more than one Media Type, you can give a unique string to Volumes with different intrinsic Media Type (Media Type = **LTO-7-8** for LTO-7 and LTO-8 types), but then those volumes will only be mounted on drives indicated with the dual type (LTO-7-8).

If you want to tie **Bacula** to using a single Storage daemon or drive, you must specify a unique Media Type for that drive. This is an important point that should be carefully understood. Note, this applies equally to Disk Volumes. If you define more than one disk Device resource in your Storage daemon's conf file, the Volumes on those two devices are in fact incompatible because one can not be mounted on the other device since they are found in different directories. For this reason, you probably should use two different Media Types for your two disk Devices (even though you might think of them as both being File types). You can find more on this subject in the Basic Volume Management chapter of this manual.

The **MediaType** specified in the Director's Storage resource, **must** correspond to the **Media Type** specified in the resource of the **Storage daemon** configuration file. This directive is required, and it is used by the Director and the Storage daemon to ensure that a Volume automatically selected from the Pool corresponds to the physical device. If a Storage daemon handles multiple devices (e.g. will write to various file Volumes on different partitions), this directive allows you to specify exactly which device.

As mentioned above, the value specified in the Director's Storage resource must agree with the value specified in the Device resource in the **Storage daemon's** configuration file. It is also an additional check so that you don't try to write data for a **LTO** onto an disk device.

Autochanger = <yes|no> If you specify **yes** for this command (the default is **no**), when you use the **label** command or the **add** command to create a new Volume, **Bacula** will also request the Autochanger Slot number. This simplifies creating database entries for Volumes in an autochanger. If you forget to specify the Slot, the autochanger will not be used. However, you may modify the Slot associated with a Volume at any time by using the **update volume** or **update slots** command in the console program. When **Autochanger** is enabled, the algorithm used by **Bacula** to search for available volumes will be modified to consider only Volumes that are known to be in the autochanger's magazine. If no **in changer** volume is found, **Bacula** will attempt recycling, pruning, ..., and if still no volume is found, **Bacula** will search for any volume whether or not in the magazine. By privileging in changer volumes, this procedure minimizes operator intervention. The default is **no**.

For the autochanger to be used, you must also specify **Autochanger=yes** in the in the Device Resource Storage Daemon's configuration file as well as other important Storage daemon configuration information. Consult the Using Autochanger chapter for the details of using autochangers. You can modify any additional resources that correspond to devices that are part of the device.

Instead of the previous **Autochanger=yes** directive, the configuration should be modified to be **Autochanger=xxx** where **xxx** is the name of the Autochanger, as documented in TechnicalReferenceDirAutochangerResource.

Maximum Concurrent Jobs = <number> where **<number>** is the maximum number of Jobs with the current Storage resource that can run concurrently. Note, this directive limits only Jobs for Jobs using this Storage daemon. Any other restrictions on the maximum concurrent jobs such as in the Director, Job, or Client resources will also apply in addition to any limit specified here. The default is set to **1**, but you may set it to a larger number. However, if you set the Storage daemon's number of concurrent jobs greater than one, we recommend that you read the waring documented under Maximum Concurrent Jobs in the Director's resource or simply turn data spooling on as documented in the Data Spooling chapter.

Maximum Concurrent Read Jobs = <number> The main purpose is to limit the number of concurrent Copy, Migration, and VirtualFull jobs so that they don't monopolize all the Storage drives causing a deadlock situation where all the drives are allocated for reading but none remain for writing. This deadlock situation can occur when running multiple simultaneous Copy, Migration, and VirtualFull jobs.

The default value is set to **0** (zero), which means there is no limit on the number of read jobs. Note, limiting the read jobs does not apply to Restore jobs, which are normally started manually. A reasonable value for this directive is one half the number of drives that the Storage resource has rounded down. Doing so will leave the same number of drives for writing and will generally avoid over committing drives and a deadlock.

AllowCompression = <yes|no> This directive is optional, and if you specify **no** (the default is **yes**), it will cause backups jobs running on this storage resource to run without client File Daemon compression. This effectively overrides compression options in FileSets used by jobs which use this storage resource. When set to **no** there will be a warning in the job log to notify that compression is disabled on this resource.

Heartbeat Interval = <time-interval> This directive is optional and if specified will cause the Director to set a keepalive interval (heartbeat) in seconds on each of the sockets it opens for the Storage resource. This value will override any specified at the Director level. It is implemented only on systems (Linux, ...) that provide the setsockopt TCP_KEEPIDLE function. The default value is **300s**.

Note: TLS Directives in the Storage resource of bacula-dir.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh** . The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

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

```
# Definition of tape storage device
Storage {
    Name = LTO-Autochanger
    Address = bacula-sd.example.com
    Password = storage_password # password for Storage daemon
    Device = "LTO-Autochanger"
    # same as Device in Storage daemon
    Media Type = LTO
    # same as MediaType in Storage daemon
    }
```

See also:

Go back to:

- *Director Resource*

- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*

Go to:

- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Autochanger Resource

The Autochanger resource supports single or multiple drive autochangers by grouping one or more Device resources into one unit called an autochanger in **Bacula** (often referred to as a "tape library" by autochanger manufacturers).

If you have an Autochanger, and you want it to function correctly, you **must** have an Autochanger resource in your Storage conf file, and your Director's Storage directives that want to use an Autochanger **must** refer to the Autochanger resource name.

Name = <Autochanger-Name> Specifies the Name of the Autochanger. This name is used in the Director's Storage resource definition to refer to the autochanger. This directive is required.

Device = <Device-name1, device-name2, . . . > Specifies the names of the Device resource or resources that correspond to the autochanger device. If you have a multiple device autochanger, you must specify multiple Device names, each one referring to a separate Device resource that contains a Drive Index specification that corresponds to the drive number base zero. You may specify multiple device names on a single line separated by commas, and/or you may specify multiple Device directives. This directive is required.

Changer Device = <name-string> The specified **<name-string>** gives the system file name of the autochanger device name. If specified in this resource, the Changer Device name is not needed in the Device resource. If it is specified in the Device resource (see above), it will take precedence over one specified in the Autochanger resource.

Changer Command = <name-string> The **<name-string>** specifies an external program to be called that will automatically change volumes as required by **Bacula**. Most frequently, you will specify the **Bacula** supplied **mtx-changer** script as follows. If it is specified here, it does not need to be specified in the Device resource. If it is also specified in the Device resource, it will take precedence over the one specified in the Autochanger resource.

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

```
Autochanger {
    Name = "LTO-8-changer"
    Device = LTO-8-1, LTO-8-2, LTO-8-3
    Changer Device = /dev/sg0
```

(continues on next page)

(continued from previous page)

```
    Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
}
Device {
    Name = "LTO-8-1"
    Drive Index = 0
    Media Type = LTO-8
    Autochanger = yes
    Archive Device = /dev/nst0
    Label Media = no
    Automatic mount = yes
    ...
}
Device {
    Name = "LTO-8-2"
    Drive Index = 1
    Media Type = LTO-8
    Autochanger = yes
    Archive Device = /dev/nst1
    Label Media = no
    Automatic mount = yes
    ...
}
Device {
    Name = "LTO-8-3"
    Drive Index = 2
    Media Type = LTO-8
    Autochanger = yes
    Autoselect = no
    Archive Device = /dev/nst2
    Label Media = no
    Automatic mount = yes
    ...
}
```

Note: It is important to include the **Autochanger = yes** directive in each Device definition that belongs to an Autochanger. A device definition should not belong to more than one Autochanger resource. Also, your Device directive in the Storage resource of the Director's conf file should have the Autochanger's resource name rather than a name of one of the Devices.

If you have a drive that physically belongs to an Autochanger but you don't want to have it automatically used when **Bacula** references the Autochanger for backups, for example, you want to reserve it for restores, you can add the directive:

```
Autoselect = no
```

to the Device resource for that drive. In that case, **Bacula** will not automatically select that drive when accessing the Autochanger. You can, still use the drive by referencing it by the Device name directly rather than the Autochanger name. An example of such a definition is shown above for the Device LTO-8-3, which will not be selected when the name LTO-8-changer is used in a Storage definition, but will be used if LTO-8-3 is used.

Specifying Slots When Labeling

If you add an **Autochanger = yes** record to the Storage resource in your Director's configuration file, the **Bacula** Console will automatically prompt you for the slot number when the Volume is in the changer when you **add** or **label** tapes for that Storage device. If your **mtx-changer** script is properly installed, **Bacula** will automatically load the correct tape during the **label** command.

You must also set **Autochanger = yes** in the Storage daemon's Device resource as we have described above in order for the autochanger to be used. See the Storage Resource in the Director's chapter and the Device Resource in the Storage daemon chapter for more details on these records.

Thus all stages of dealing with tapes can be totally automated. It is also possible to set or change the Slot using **update** the command in the Console and selecting **Volume Parameters** to update.

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

If your autochanger has barcode labels, you can label all the Volumes in your autochanger one after another by using the **label barcodes** command. For each tape in the changer containing a barcode, **Bacula** will mount the tape and then label it with the same name as the barcode. An appropriate Media record will also be created in the catalog. Any barcode that begins with the same characters as specified on the **"CleaningPrefix=xxx** command, will be treated as a cleaning tape, and will not be labeled.

For example with:

```
Pool {
    Name ...
    Cleaning Prefix = "CLN"
    }
```

any slot containing a barcode of `CLNxxxx` will be treated as a cleaning tape and will not be mounted.

Note: Volumes must be pre-labeled to be automatically used in the autochanger during a backup. If you do not have a barcode reader, this is done manually (or via a script).

Changing Cartridges

If you wish to insert or remove cartridges in your autochanger or you manually run the **mtx** program, you must first tell **Bacula** to release the autochanger by doing:

```
unmount
```

Change cartridges and/or run **mtx**

```
mount
```

If you do not do the **unmount** before making such a change, **Bacula** will become completely confused about what is in the autochanger and may stop function because it expects to have exclusive use of the autochanger while it has the drive mounted.

Dealing with Multiple Magazines

If you have several magazines or if you insert or remove cartridges from a magazine, you should notify **Bacula** of this. By doing so, **Bacula** will as a preference, use Volumes that it knows to be in the autochanger before accessing Volumes that are not in the autochanger. This prevents unneeded operator intervention.

If your autochanger has barcodes (machine readable tape labels), the task of informing **Bacula** is simple. Every time, you change a magazine, or add or remove a cartridge from the magazine, simply do

```
unmount
```

Remove magazine

Insert new magazine

```
update slots
mount
```

in the Console program. This will cause **Bacula** to request the autochanger to return the current Volume names in the magazine. This will be done without actually accessing or reading the Volumes because the barcode reader does this during inventory when the autochanger is first turned on. **Bacula** will ensure that any Volumes that are currently marked as being in the magazine are marked as no longer in the magazine, and the new list of Volumes will be marked as being in the magazine. In addition, the Slot numbers of the Volumes will be corrected in **Bacula**'s catalog if they are incorrect (added or moved).

If you do not have a barcode reader on your autochanger, you have several alternatives.

1. You can manually set the Slot and InChanger flag using the **update volume** command in the Console (quite painful).
2. You can issue a

   ```
   update slots scan
   ```

   command that will cause **Bacula** to read the label on each of the cartridges in the magazine in turn and update the information (Slot, InChanger flag) in the catalog. This is quite effective but does take time to load each cartridge into the drive in turn and read the Volume label.
3. You can modify the script so that it simulates an autochanger with barcodes. See below for more details.

Simulating Barcodes in Autochanger

You can simulate barcodes in your autochanger by making the **mtx-changer** script return the same information that an autochanger with barcodes would do. This is done by commenting out the one and only line in the **list**) case, which is:

```
:math: cat ${TMPFILE} | grep " Storage Element [0-9]*:.*Full" | awk "{print \$3 \$4}" |␣
↪sed "s/Full *\(:VolumeTag=\)*//"
```

by putting a # in column one of that line, or by simply deleting it. Then in its place add a new line that prints the contents of a file. For example:

```
cat /etc/bacula/changer.volumes
```

Be sure to include a full path to the file, which can have any name. The contents of the file must be of the following format:


```
1:Volume1
2:Volume2
3:Volume3 ...
```

Where the 1, 2, 3 are the slot numbers and Volume1, Volume2, ... are the Volume names in those slots. You can have multiple files that represent the Volumes in different magazines, and when you change magazines, simply copy the contents of the correct file into your **/etc/bacula/changer.volumes** file. There is no need to stop and start **Bacula** when you change magazines, simply put the correct data in the file, then run **update slots** the command, and your autochanger will appear to **Bacula** to be an autochanger with barcodes.

The Full Form of the Update Slots Command

If you change only one cartridge in the magazine, you may not want to scan all Volumes, so the **update slots** command (as well as the **update slots scan** command) has the additional form:

```
update slots=n1,n2,n3-n4, ...
```

where the keyword **scan** can be appended or not. The n1,n2, ... represent Slot numbers to be updated and the form n3-n4 represents a range of Slot numbers to be updated (e.g. 4-7 will update Slots 4,5,6, and 7).

This form is particularly useful if you want to do a scan (time expensive) and restrict the update to one or two slots.

For example, the command:

```
update slots=1,6 scan
```

will cause **Bacula** to load the Volume in Slot 1, read its Volume label and update the Catalog. It will do the same for the Volume in Slot 6. The command:

```
update slots=1-3,6
```

will read the barcoded Volume names for slots 1,2,3 and 6 and make the appropriate updates in the Catalog. If you don't have a barcode reader or have not modified the **mtx-changer** script as described above, the above command will not find any Volume names so will do nothing.

Testing Autochanger and Adapting mtx-changer script

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 **/opt/bacula/scripts/mtx-changer**):

Make sure Bacula is not running.

/opt/bacula/scripts/mtx-changer /dev/sg0 list 0 /dev/nst0 0 This command should print:

```
1:
2:
3:
...
```

or one number per line for each slot that is occupied in your changer, and the number should be terminated by a colon (**:**). If your changer has barcodes, the barcode will follow the colon. If an error message is printed, you must resolve the problem (e.g. try a different SCSI control device name if **/dev/sg0** is incorrect). For example, on FreeBSD systems, the autochanger SCSI control device is generally **/dev/pass2**.

/opt/bacula/scripts/mtx-changer /dev/sg0 listall 0 /dev/nst0 0 This command should print:

```
Drive content:      D:Drive num:F:Slot loaded:Volume Name
D:0:F:2:vol2        or D:Drive num:E
D:1:F:42:vol42
D:3:E

Slot content:
S:1:F:vol1          S:Slot num:F:Volume Name
S:2:E               or S:Slot num:E
S:3:F:vol4

Import/Export tray slots:
I:10:F:vol10        I:Slot num:F:Volume Name
I:11:E              or I:Slot num:E
I:12:F:vol40
```

/opt/bacula/scripts/mtx-changer /dev/sg0 transfer This command should transfer a volume from source (1) to destination (2)

/opt/bacula/scripts/mtx-changer /dev/sg0 slots This command should return the number of slots in your autochanger.

/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0 If a tape is loaded from slot 1, this should cause it to be unloaded.

/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0 Assuming you have a tape in slot 3, it will be loaded into drive (0).

/opt/bacula/scripts/mtx-changer /dev/sg0 loaded 0 /dev/nst0 0 It should print "3" Note, we have used an "illegal" slot number 0. In this case, it is simply ignored because the slot number is not used. However, it must be specified because the drive parameter at the end of the command is needed to select the correct drive.

/opt/bacula/scripts/mtx-changer/dev/sg0unload3 /dev/nst00 will unload the tape into slot 3.

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. The only remaining area of problems will be if your autoloader needs some time to get the tape loaded after issuing the command. After the **mtx-changer** script returns, **Bacula** will immediately rewind and read the tape. If **Bacula** gets rewind I/O errors after a tape change, you will probably need to insert a **sleep 20** after the **mtx** command, but be careful to exit the script with a zero status by adding **exit 0** after any additional commands you add to the script. This is because **Bacula** checks the return status of the script, which should be zero if all went well.

You can test whether or not you need a **sleep** by putting the following commands into a file and running it as a script:

```
#!/bin/sh
/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0
mt -f/dev/st0 rewind
mt -f /dev/st0 weof
```

If the above script runs, you probably have no timing problems. If it does not run, start by putting a **sleep 30** or possibly a **sleep 60** in the script just after the **mtx-changer** load command. If that works, then you should move the sleep into the actual **mtx-changer** script so that it will be effective when **Bacula** runs.

A second problem that comes up with a small number of autochangers is that they need to have the cartridge ejected before it can be removed. If this is the case, the **load 3** will never succeed regardless of how long you wait. If this seems to be your problem, you can insert an eject just after the unload so that the script looks like:


```
#!/bin/sh
/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
mt -f /dev/st0 offline
/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0
mt -f /dev/st0 rewind
mt -f /dev/st0 weof
```

Obviously, if you need the **offline** command, you should move it into the **mtx-changer** script ensuring that you save the status of the **mtx** command or always force an **exit 0** from the script, because **Bacula** checks the return status of the script.

As noted earlier, there are several scripts in **<bacula-source>/examples/devices** that implement the above features, so they may be a help to you in getting your script to work.

If **Bacula** complains "Rewind error on **/dev/nst0**. ERR=Input/output error." you most likely need more sleep time in your **mtx-changer** before returning to **Bacula** after a load command has been completed.

Using the Autochanger

Let's assume that you have properly defined the necessary Storage Daemon Device records, and you have added the **Autochanger = yes** record to the Storage 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:

```
/opt/bacula/bin/bconsole
Connecting to Director bacula-dir:9101
1000 OK: 10002 bacula-dir Version: 16.0.5 (05 April 2023)
Enter a period to cancel a command.
*label
```

it will then print something like:

```
Using Catalog "BaculaCatalog"
The defined Storage resources are:
        1: TapeAutochanger
        2: DiskAutochanger
Select Storage resource (1-2):1
```

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

```
Connecting to Storage daemon DiskAutochanger at baculasd:9103 ...
Enter autochanger drive[0]:
Enter new Volume name: LTO-8-Tape1
Enter slot (0 or Enter for none): 1
```

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

```
Defined Pools:
    1: Tape-pool
```

(continues on next page)

(continued from previous page)

```
    2: DiskBackup365d
Select the Pool (1-2): 1
```

I select the Tape-pool 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 TapeAutochanger at baculasd: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 Catalog "BaculaCatalog"
    Defined Pools:
        1: Tape-pool
        2: DiskBackup365d
    Select the Pool (1-2): 1
    +-------+-------------+--------+---------+-------+--------+----------+-------+------+
    | MedId | VolName     | MedTyp | VolStat | Bites | LstWrt | VolReten | Recyc | Slot |
    +-------+-------------+--------+---------+-------+--------+----------+-------+------+
    | 1     | LTO-8-Tape1 | LTO-8  | Append  | 0     | 0      | 30672000 | 0     | 1    |
    | 2     | LTO-8-Tape1 | LTO-8  | Append  | 0     | 0      | 30672000 | 0     | 2    |
    | 3     | LTO-8-Tape1 | LTO-8  | Append  | 0     | 0      | 30672000 | 0     | 3    |
    | ...                                                                            |
    +-------+-------------+--------+---------+-------+--------+----------+-------+------+
```

Barcode Support

Bacula provides barcode support with two Console commands, **label barcodes** and **update slots**.

The **label barcodes** will cause **Bacula** to read the barcodes of all the cassettes that are currently installed in the magazine (cassette holder) using the **mtx-changer list** command. Each cassette is mounted in turn and labeled with the same Volume name as the barcode.

The **update slots** command will first obtain the list of cassettes and their barcodes from **mtx-changer**. Then it will find each volume in turn in the catalog database corresponding to the barcodes and set its slot to correspond to the value just read. If the Volume is not in the catalog, then nothing will be done. This command is useful for synchronizing **Bacula** with the current magazine in case you have changed magazines or in case you have moved cassettes from one slot to another. If the autochanger is empty, nothing will be done.

The **Cleaning Prefix** statement can be used in the Pool resource to define a Volume name prefix, which if it matches that of the Volume (barcode) will cause that Volume to be marked with a VolStatus of **Cleaning**. This will prevent **Bacula** from attempting to write on the Volume.

Use bconsole to Display Autochanger Content

The **status slots storage=xxx** command displays autochanger content.

```
 Slot |  Volume Name    |  Status  |      Type         |    Pool        |  Loaded |
------+-----------------+----------+-------------------+----------------+---------|
    1 |           00001 |   Append |  DiskChangerMedia |        Default |    0    |
    2 |           00002 |   Append |  DiskChangerMedia |        Default |    0    |
    3*|           00003 |   Append |  DiskChangerMedia |        Scratch |    0    |
    4 |                 |          |                   |                |    0    |
```

If you see a "*" near the slot number, you have to run **update slots** command to synchronize autochanger content with your catalog.

Setting Up a Tape Autochanger Using Bweb

Setting Up a Tape Autochanger Using Bweb

Bacula Autochanger Interface

Bacula calls the autochanger script that you specify on the **Changer Command** statement. Normally this script will be the **mtx-changer** script that we provide, but it can in fact be any program. The only requirement for the script is that it must understand the commands that **Bacula** uses, which are **loaded**, **load**, **unload**, **list**, and **slots**. In addition, each of those commands must return the information in the precise format as specified below:

Currently the changer commands used are:

loaded returns number of the slot that is loaded, base 1, in the drive or 0 if the drive is empty.

load loads a specified slot (some autochangers require a 30 second pause after this command) into the drive.

upload unloads the device (returns cassette to its slot).

list returns one line for each cassette in the autochanger in the format <slot>:<barcode>. Where the is the non-zero integer representing the slot number, and is the barcode associated with the cassette if it exists and if you autoloader supports barcodes. Otherwise the barcode field is blank.

slot returns total number of slots in the autochanger.

Bacula checks the exit status of the program called, and if it is zero, the data is accepted. If the exit status is non-zero, **Bacula** will print an error message and request the tape be manually mounted on the drive.

Autochanger Support

Bacula provides autochanger support for reading and writing tapes. In order to work with an autochanger, **Bacula** requires a number of things, each of which is explained in more detail after this list:

- A script that actually controls the autochanger according to commands sent by **Bacula**. We furnish such a script that works with found **mtx** in most linux distributions.
- That each Volume (tape) to be used must be defined in the Catalog and have a Slot number assigned to it so that **Bacula** knows where the Volume is in the autochanger. This is generally done with the **label** command, but can also done after the tape is labeled using the **update slots** command. See below for more details. You must pre-label the tapes manually before using them.
- Modifications to your Storage daemon's Device configuration resource to identify that the device is a changer, as well as a few other parameters.
- You should also modify your Storage resource definition in the Director's configuration file so that you are automatically prompted for the Slot when labeling a Volume.
- You need to ensure that your Storage daemon (if not running as root) has access permissions to both the tape drive and the control device.
- You need to have **Autochanger = yes** in your Storage resource in your **bacula-dir.conf** file so that you will be prompted for the slot number when you label Volumes.

The Autochanger resource permits you to group Device resources thus creating a multi-drive autochanger. If you have a tape library with multiple drives, you **must** use this resource.

Bacula uses its own **mtx-changer** script to interface with a program that actually does the tape changing. Thus in principle, **mtx-changer** can be adapted to function with any autochanger program, or you can call any other script or program. The current version of **mtx-changer** works with the **mtx** program. FreeBSD users have provided a script in the **examples/autochangers** directory that allows **Bacula** to use the **chio** program.

Bacula also supports autochangers with barcode readers. This support includes two Console commands: **label barcodes** and **update slots** . For more details on these commands, see the "Barcode Support" section below.

Current **Bacula** autochanger support does not include cleaning, stackers, or silos. Stackers and silos are not supported because **Bacula** expects to be able to access the slots randomly. However, if you are very careful to setup **Bacula** to access the Volumes in the autochanger sequentially, you may be able to make **Bacula** work with stackers (gravity feed and such).

In principle, **mtx** if will operate your changer correctly, then it is just a question of adapting the **mtx-changer** script (or selecting one already adapted) for proper interfacing.

If you are having troubles, please use the **auto** command in the **btape** program to test the functioning of your autochanger with **Bacula**. When **Bacula** is running, please remember that for many distributions (e.g. FreeBSD, Debian, ...) the Storage daemon runs as **bacula.tape** rather than **root.root**, so you will need to ensure that the Storage daemon has sufficient permissions to access the autochanger.

Some users have reported that the the Storage daemon blocks under certain circumstances in trying to mount a volume on a drive that has a different volume loaded. As best we can determine, this is simply a matter of waiting a bit. The drive was previously in use writing a Volume, and sometimes the drive will remain `BLOCKED` for a good deal of time (up to 7 minutes on a slow drive) waiting for the cassette to rewind and to unload before the drive can be used with a different Volume.

Knowing What SCSI Devices You Have

Under Linux, you can

```
cat /proc/scsi/scsi
```

to see what SCSI devices you have available. You can also:

```
cat /proc/scsi/sg/device_hdr /proc/scsi/sg/devices
```

to find out how to specify their control address (**/dev/sg0** for the first, **/dev/sg1** for the second, ...) on the **Changer Device = Bacula** directive.

You can also use the excellent **lsscsi** tool.

```
$ lsscsi -g
[1:0:2:0]     tape    SEAGATE    ULTRIUM06242-XXX    1619    /dev/st0    /dev/sg9
[1:0:14:0]    mediumx     STK         L180           0315    /dev/sch0   /dev/sg10
[2:0:3:0]     tape         HP    Ultrium 3-SCSI      G24S    /dev/st1    /dev/sg11
[3:0:0:0]     enclosu      HP        A6255A          HP04        -       /dev/sg3
[3:0:1:0]     disk     HP 36.4G    ST336753FC        HP00    /dev/sdd    /dev/sg4
```

For more detailed information on what SCSI devices you have please see the Linux SCSI Tricks problems section of the of the Tape Testing section of the Bacula Enterprise Problems Resolution Guide.

Under FreeBSD, you can use:

```
camcontrol devlist
```

To list the SCSI devices as well as the **/dev/passn** that you will use on the **Bacula Changer Device** directive.

Check that your Storage daemon has permission to access this device.

The tip for FreeBSD users: on reboot **Bacula** will NOT have permission to control the device **/dev/pass0** (assuming this is your changer device). To get around this just edit the **/etc/devfs.conf** file and add the following to the bottom:

```
own pass0 root:bacula
perm pass0 0666
own nsa0.0 root:bacula
perm nsa0.0 0666
```

This gives the bacula group permission to write to the **nsa0.0** device too just to be on the safe side. To bring these changes into effect just run:

```
/etc/rc.d/devfs restart
```

Basically this will stop you having to manually change permissions on these devices to make **Bacula** work when operating the AutoChanger after a reboot.

The mtx-changer script

Read the sections below so that you understand how autochangers work with **Bacula**. Although we supply a default **mtx-changer** script, your autochanger may require some additional changes.

Slots

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.

Bacula will not automatically use a Volume in your autochanger unless it is labeled and the slot number is stored in the catalog and the Volume is marked as InChanger. This is because it must know where each volume is (slot) to be able to load the volume.

For each Volume in your changer, you will, using the Console program, assign a slot. This information is kept in **Bacula's** catalog 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. When doing a **mount** command on an autochanger, you must specify which slot you want mounted. If the drive is loaded with a tape from another slot, it will unload it and load the correct tape, but normally, no tape will be loaded because an **unmount** command causes **Bacula** to unload the tape in the drive.

You can check if the Slot number and InChanger flag are set by doing a:

```
list Volumes
```

in the Console program.

Multiple Devices

Some autochangers have more than one read/write device (drive). The Autochanger resource permits you to group Device resources, where each device represents a drive. The Director may still reference the Devices (drives) directly, but doing so, bypasses the proper functioning of the drives together. Instead, the Director (in the Storage resource) should reference the Autochanger resource name. Doing so permits the Storage daemon to ensure that only one drive uses the **mtx-changer** script at a time, and also that two drives don't reference the same Volume.

Multi-drive requires the use of the **Drive Index** directive in the resource of the Storage daemon's configuration file. Drive numbers or the Device Index are numbered beginning at zero (0), which is the default. To use the second Drive in an autochanger, you need to define a second resource and set the ****Drive Index**** to 1 for that device. In general, the second device will have the same **Changer Device** (control channel) as the first drive, but a different **Archive Device**.

As a default, **Bacula** jobs will prefer to write to a Volume that is already mounted and available for writing. If you have a multiple drive autochanger and you want **Bacula** to write to more than one Volume in the same Pool at the same time, you will probably set MaximumConcurrentJobs to 1 in each tape device. This will cause the Storage daemon to maximize the use of drives.

Device Configuration Records

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

These four records, permitted in resources, are described in detail below. Note, however, that the **Changer Device** and the **Changer Command** directives are not needed in the resource if they are present in the **Autochanger** resource.

Autochanger = <yes|no> The **Autochanger** 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** . Note, some of the more advanced autochangers will locate the changer device on **/dev/sg1**. Such devices typically have several drives and a large number of tapes.

On FreeBSD systems, the changer device will typically be on **/dev/pass0** through **/dev/passn**.

On Solaris, the changer device will typically be some file under **/dev/rdsk**.

Ensure that your Storage daemon has permission to access this device.

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
%d = changer drive index base 0
%f = Client's name
%i = JobId
%j = Job name
%l = archive control channel name
%o = command (loaded, load, or unload)
%s = Slot base 0
%S = Slot base 1
%v = Volume name
%V = Volume name from dcr->VolCatInfo first
```

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

```
Changer Command = "/opt/bacula/scripts/mtx-changer
```

Where you will need to adapt the **/opt/bacula** to be the actual path on your system where the **mtx-changer** script resides. 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 **5 minutes**. 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.

Drive Index = <number> This record allows you to tell **Bacula** to use the second or subsequent drive in an autochanger with multiple drives. Since the drives are numbered from zero, the second drive is defined by

```
Drive Index = 1
```

To use the second drive, you need a second Device resource definition in the **Bacula** configuration file. See the *Multiple Drive* section above in this chapter for more information.

In addition, for proper functioning of the Autochanger, you must define an Autochanger resource.

Note: If your tape hardware and operating system are relatively new (<10 years), they will cooperate smoothly.

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*

Go to:

- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Pool Resource

The Pool resource defines the set of storage Volumes (tapes or files) to be used by **Bacula** to write the data. By configuring different Pools, you can determine which set of Volumes (media) receives the backup data. This permits, for example, to store all full backup data on one set of Volumes and all incremental backups on another set of Volumes. Alternatively, you could assign a different set of Volumes to each machine that you backup. This is most easily done by defining multiple Pools.

Another important aspect of a Pool is that it contains the default attributes (Maximum Jobs, Retention Period, Recycle flag, etc.) that will be given to a Volume when it is created. This avoids the need for you to answer a large number of questions when labeling a new Volume. Each of these attributes can later be changed on a Volume by Volume basis using the **update** command in the console program. Note that you must explicitly specify which Pool **Bacula** is to use with each Job. **Bacula** will not automatically search for the correct Pool.

Most often in **Bacula** installations all backups for all machines (Clients) go to a single set of Volumes. In this case, you will probably only use the **Default** Pool. If your backup strategy calls for you to mount a different tape each day, you will probably want to define a separate Pool for each day. For more information on this subject, see the Backup Strategies chapter.

To use a Pool, there are three distinct steps. First the Pool must be defined in the Director's configuration file. Then the Pool must be written to the Catalog database. This is done automatically by the Director each time that it starts, or alternatively can be done using the **create** command in the console program. Finally, if you change the Pool definition in the Director's configuration file and restart **Bacula**, the pool will be updated alternatively you can use the **update pool** console command to refresh the database image. It is this database image rather than the Director's resource image that is used for the default Volume attributes. Note, for the pool to be automatically created or updated, it must be explicitly referenced by a Job resource.

Next the physical media must be labeled. The labeling can either be done with the **label** command in the **bconsole** program or using the **btape** program. The preferred method is to use the **label** command in the **bconsole** program.

Finally, you must add Volume names (and their attributes) to the Pool. For Volumes to be used by **Bacula** they must be of the same **Media Type** as the archive device specified for the job (i.e. if you are going to back up to a **LTO** device, the Pool must have LTO volumes defined since 8mm volumes cannot be mounted on a **LTO** 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 bconsole program to label the Volumes, they will automatically be added to the Pool, so this last step is not normally required.

It is also possible to add Volumes to the database without explicitly labeling the physical volume. This is done with the **add** bconsole command.

As previously mentioned, each time **Bacula** starts, it scans all the Pools associated with each Catalog, and if the database record does not already exist, it will be created from the Pool Resource definition. **Bacula** probably should do an **update pool** if you change the Pool definition, but currently, you must do this manually using the **update pool** command in the bconsole program.

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

Pool Start of the Pool resource. There must be at least one Pool resource defined.

Name = <name> The name of the pool. For most applications, you will use the default pool name **Default**. This directive is required.

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

This directive is only respected in case of volumes automatically created by **Bacula**. If you add volumes to a pool manually with the **label** command, it is possible to have more volumes in a pool than specified by **Maximum Volumes**.

Maximum Pool Bytes = <size> This directive specifies the total maximum size of Volumes in the Pool.

Note: Available as of Version 10.0.

This directive is optional, if omitted or set to zero, any size will be permitted. The current size of the Pool can be displayed with **llist pool** command. Note that the size displayed is not necessarily the sum of the size of all Volumes; it represents the effective amount of data, which can be much larger than the Volume size in case of Global Endpoint Deduplication Volumes, or smaller than the apparent file size in case of Aligned Volumes.

All types of volumes with any status are counted in the Pool size.

To exclude purged Volumes from the quota, the **RecyclePool** directive can be used to assign volumes to a Scratch pool once they are no longer containing valid data, or Volumes can be truncated using the **ActionOnPurge** functionality.

There are cases Bacula can not adhere to the Pool size limitation in a strict manner:

- For Backups stored on Aligned or Deduplication Volumes, a single Volume write can represent very large amounts of data, which may lead to temporarily exceeding the configured sizes.
- As the Pool Size check is done by the storage Device handler of the Storage Daemon, concurrently writing to a Pool with this setting by different Storage Devices may result in exceeding the set limit. After Volumes have been changed, the then-correct Pool size will be used.

These limitations may be resolved in a later version of Bacula.

Pool Type = Backup This directive defines the pool type. It is required and the only available value is **Backup**.

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

StorageGroupPolicy = <Storage Group Policy Name> Storage Group Policy determines how Storage resources (from the 'Storage' directive) are being chosen from the Storage list. If no StoragePolicy is specified Bacula always tries to use first available Storage from the provided list. See DirectorJobStorageGroupPolicy for more information.

Use Volume Once = <yes|no> This directive if set to **yes** specifies that each volume is to be used only once. This is most useful when the Media is a file and you want a new file for each backup that is done. The default is **no** (i.e. use volume any number of times). This directive will most likely be phased out (deprecated), so you are recommended to use **MaximumVolumeJobs = 1** instead.

The value defined by this directive in the **bacula-dir.conf** file is the default value used when a Volume is created. Once the volume is created, changing the value in the **bacula-dir.conf** file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in bconsole.

See the notes below under **MaximumVolumeJobs** concerning using this directive with multiple simultaneous jobs.

Maximum Volume Jobs = <positive-integer> This directive specifies the maximum number of Jobs that can be written to the Volume. If you specify **0** (the default), there is no limit. Otherwise, when the number of Jobs backed up to the Volume equals the Volume will be marked . When the Volume is marked it can no longer be used for appending Jobs, much like the status. A Volume that is marked or can be recycled if recycling is enabled, and thus used again. By setting **MaximumVolumeJobs** to **1**, you get the same effect as setting **UseVolumeOnce=yes**.

The value defined by this directive in the **bacula-dir.conf** file is the default value used when a Volume is created. Once the volume is created, changing the value in the **bacula-dir.conf** file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

If you are running multiple simultaneous jobs, this directive may not work correctly because when a drive is reserved for a job, this directive is not taken into account, so multiple jobs may try to start writing to the Volume. At some point, when the Media record is updated, multiple simultaneous jobs may fail since the Volume can no longer be written.

Maximum Volume Files = <positive-integer> This directive specifies the maximum number of files that can be written to the Volume. If you specify **0**, (the default), there is no limit. Otherwise, when the number of files written to the Volume equals the Volume will be marked . When the Volume is marked it can no longer be used for appending Jobs, much like the status, but it can be recycled if recycling is enabled and thus used again. This value is checked and the status is set only at the end of a job that writes to the particular volume.

The value defined by this directive in the **bacula-dir.conf** file is the default value used when a Volume is created. Once the volume is created, changing the value in the **bacula-dir.conf** file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in bconsole.

Maximum Volume Bytes = <size> This directive specifies the maximum number of bytes that can be written to the Volume. If you specify zero (**0**,the default), there is no limit except the physical size of the Volume. Otherwise, when the number of bytes written to the Volume equals the Volume will be marked . When the Volume is marked it can no longer be used for appending Jobs, but it can be recycled if recycling is enabled, and thus the Volume can be re-used after recycling. The size specified is checked just before each block is written to the Volume and if the Volume size would exceed the specified Maximum Volume Bytes the status will be set and the Job will request the next available Volume to continue.

This directive is particularly useful for restricting the size of disk volumes, and w ill work correctly even in the case of multiple simultaneous jobs writing to the volume.

The value defined by this directive in the **bacula-dir.conf** file is the default value used when a Volume is created. Once the volume is created, changing the value in the **bacula-dir.conf** file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

Volume Use Duration = <time-period-specification> The Volume Use Duration directive defines the time period that the Volume can be written beginning from the time of first data write to the Volume. If the time-period specified **0** (the default), the Volume can be written indefinitely. Otherwise, the next time a job runs that wants to access this Volume, and the time period from the first write to the volume (the first Job written) exceeds the time-period-specification, the Volume will be marked, which means that no more Jobs can be appended to the Volume, but it may be recycled if recycling is enabled. Using the command **status dir** applies algorithms similar to running jobs, so during such a command, the Volume status may also be changed. Once the Volume is recycled, it will be available for use again.

You might use this directive, for example, if you have a Volume used for Incremental backups, and Volumes used for Weekly Full backups. Once the Full backup is done, you will want to use a different Incremental Volume. This can be accomplished by setting the Volume Use Duration for the Incremental Volume to six days. I.e. it will be used for the 6 days following a Full save, then a different Incremental volume will be used. Be careful about setting the duration to short periods such as 23 hours, or you might experience problems of **Bacula** waiting for a tape over the weekend only to complete the backups Monday morning when an operator mounts a new tape.

The VolumeUseDuration is checked and the **Used** status is set only at the end of a job that writes to the particular volume, which means that even though the use duration may have expired, the catalog entry will not be updated until the next job that uses this volume is run. This directive is not intended to be used to limit volume sizes and may not work as expected (i.e. will fail jobs) if the use duration expires while multiple simultaneous jobs are writing to the volume.

Note: The value defined by this directive in the **bacula-dir.conf** file is the default value used when a Volume is created. Once the volume is created, changing the value in the **bacula-dir.conf** file will not change what is stored for the Volume. To change the value for an existing Volume you must use update volume command.

Catalog Files = <yes|no> This directive defines whether or not you want the names of the files that were saved to be put into the catalog. The default is **yes**. The advantage of specifying **Catalog Files=No** is that you will have a significantly smaller Catalog database. The disadvantage is that you will not be able to produce a Catalog listing of the files backed up for each Job (this is often called Browsing). Also, without the File entries in the catalog, you will not be able to use the bconsole **restore** command nor any other command that references file entries.

AutoPrune = <yes|no> If AutoPrune is set to **yes** (default), **Bacula** 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 **VolumeRetention** directive defines the longest amount of time that **Bacula** will keep records associated with the Volume in the Catalog database after the end time of each Job written to the Volume. When this time period expires, and if **AutoPrune** is set to **yes**, **Bacula** may prune (remove) Job records that are older than the specified Volume Retention period if it is necessary to free up a Volume. Note, it is also possible for all the Job and File records to be pruned before the Volume Retention period if Job and File Retention periods are

configured to a lower value. In that case the Volume can then be marked Pruned and subsequently recycled prior to expiration of the Volume Retention period.

Recycling will not occur until it is absolutely necessary to free up a volume (i.e. no other writable volume exists). All File records associated with pruned Jobs are also pruned. The time may be specified as seconds, minutes, hours, days, weeks, months, quarters, or years. The **VolumeRetention** is applied independently of the **JobRetention** and the **FileRetention** periods defined in the Client resource. This means that all the retention periods are applied in turn and that the shorter period is the one that effectively takes precedence. Note, that when the **VolumeRetention** period has been reached, and it is necessary to obtain a new volume, **Bacula** will prune both the Job and the File records. And the inverse is also true that if all the Job and File records that refer to a Volume were already pruned, then the Volume may be recycled regardless of its retention period. Pruning may also occur during a **status dir** command because it uses similar algorithms for finding the next available Volume.

It is important to know that when the **VolumeRetention** period expires, or all the Job and File records have been pruned that refer to a Volume, **Bacula** does not automatically recycle a Volume. It attempts to keep the Volume data intact as long as possible before over writing the Volume.

By defining multiple Pools with different Volume Retention periods, you may effectively have a set of tapes that is recycled weekly, another Pool of tapes that is recycled monthly and so on. However, one must keep in mind that if your **VolumeRetention** period is too short, it may prune the last valid Full backup, and hence until the next Full backup is done, you will not have a complete backup of your system, and in addition, the next Incremental or Differential backup will be promoted to a Full backup. As a consequence, the minimum **VolumeRetention** period should be at twice the interval of your Full backups. This means that if you do a Full backup once a month, the minimum Volume retention period should be two months.

The default Volume retention period is **365 days**, and either the default or the value defined by this directive in the **bacula-dir.conf** file is the default value used when a Volume is created. Once the volume is created, changing the value in the **bacula-dir.conf** file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

To disable the **VolumeRetention** feature, it is possible to set the directive to **0**. When disabled, the pruning will be done only on the **JobRetention** directive and the "ExpiresIn" information available in the **list volume** output is not available.

Cache Retention = <time-period-specification> The **Cache Retention** directive defines the longest amount of time that bacula will keep Parts associated with the Volume in the Storage daemon Cache directory after a successful upload to the Cloud. When this time period expires, and if the **cloud prune** command is issued, **Bacula** may prune (remove) Parts that are older than the specified **Cache Retention** period.

Note: It is also possible for all the Parts to be removed before the Cache Retention period is reached. In that case the **cloud truncate** command must be used.

Action On Purge = <Truncate> The directive **ActionOnPurge = Truncate** instructs **Bacula** to permit the Volume to be truncated after it has been purged.

Note: The ActionOnPurge is a bit misleading since the volume is not actually truncated when it is purged, but is enabled to be truncated. The actual truncation is done with the **truncate** command.

To actually truncate a Volume, you must first set the **ActionOnPurge** to Truncate in the Pool, then you must ensure that any existing Volumes also have this information in them, by doing an **update volumes** comand. Finally, after the Volume has been purged, you may then truncate it. It is useful to prevent disk based volumes from consuming too much space. See below for more details of how to ensure Volumes are truncated after being purged.

First set the Pool to permit truncation.

```
Pool {
    Name = Default
```

(continues on next page)

(continued from previous page)

```
    Action On Purge = Truncate
    ...
}
```

Then assuming a Volume has been Purged, you can schedule a truncate operation at the end of your CatalogBackup job like in this example:

```
Job {
    Name = CatalogBackup
    ...
    RunScript {
        RunsWhen = After
        RunsOnClient = No
        Console = "truncate volume allpools storage=File"
    }
}
```

ScratchPool = <pool-resource-name> This directive permits specifying a specific scratch Pool to be used for the Job. This pool will replace the default scratch pool named *Scratch* for volume selection. For more information about scratch pools, see *Scratch Pool* section of this manual. This directive is useful when using multiple storage devices that share the same MediaType or when you want to dedicate volumes to a particular set of pools.

RecyclePool = <pool-resource-name> This directive defines to which pool the Volume will be placed (moved) when it is recycled. Without this directive, a Volume will remain in the same pool when it is recycled. With this directive, it will be moved automatically to any existing pool during a recycle. This directive is probably most useful when defined in the Scratch pool, so that volumes will be recycled back into the Scratch pool. For more, see the *Scratch Pool* section.

Although this directive is called RecyclePool, the Volume in question is actually moved from its current pool to the one you specify on this directive when **Bacula** prunes the Volume and discovers that there are no records left in the catalog and hence marks it as **Purged**.

Recycle = <yes|no> This directive specifies whether or not Purged Volumes may be recycled. If it is set to **yes** (the default) and **Bacula** needs a volume but finds none that are appendable, it will search for and recycle (reuse) Purged Volumes (i.e. volumes with all the Jobs and Files expired and thus deleted from the Catalog). If the Volume is recycled, all previous data written to that Volume will be overwritten. If Recycle is set to **no**, the Volume will not be recycled, and hence, the data will remain valid. If you want to reuse (re-write) the Volume, and the recycle flag is no (0 in the catalog), you may manually set the recycle flag (**update** command) for a Volume to be reused.

Note: The value defined by this directive in the **bacula-dir.conf** file is the default value used when a Volume is created. Once the volume is created, changing the value in the **bacula-dir.conf** file will not change what is stored for the Volume. To change the value for an existing Volume you must use the **update** command in the Console.

When all Job and File records have been pruned or purged from the catalog for a particular Volume, if that Volume is marked as Full or Used, it will then be marked as Purged. Only Volumes marked as Purged will be considered to be converted to the Recycled state if the **Recycle** directive is set to **yes**.

Recycle Oldest Volume = <yes|no> This directive instructs the Director to search for the oldest used Volume in the Pool when another Volume is requested by the Storage Daemon and none are available. The catalog is then **pruned** respecting the retention periods of all Files and Jobs written to this Volume. If all Jobs are pruned (i.e. the volume is Purged), then the Volume is recycled and will be used as the next Volume to be written. This directive respects any Job, File, or Volume retention periods that you may have specified, and as such it is much better to use this directive than the **PurgeOldestVolume**.

This directive can be useful if you have a fixed number of Volumes in the Pool and you want to cycle through them and

you have specified the correct retention periods.

However, if you use this directive and have only one Volume in the Pool, you will immediately recycle your Volume if you fill it and **Bacula** needs another one. Thus your backup will be totally invalid. Please use this directive with care. The default is **no**.

Recycle Current Volume = <yes|no> If **Bacula** needs a new Volume, this directive instructs to Prune the volume respecting the Job and File retention periods. If all Jobs are pruned (i.e. the volume is Purged), then the Volume is recycled and will be used as the next Volume to be written. This directive respects any Job, File, or Volume retention periods that you may have specified, and thus it is much better to use it rather than the Purge Oldest Volume directive.

This directive can be useful if you have a fixed number of Volumes in the Pool, you want to cycle through them, and you have specified retention periods that prune Volumes before you have cycled through the Volume in the Pool.

However, if you use this directive and have only one Volume in the Pool, you will immediately recycle your Volume if you fill it and **Bacula** needs another one. Thus your backup will be totally invalid. Please use this directive with care. The default is **no**.

Purge Oldest Volume = <yes|no> This directive instructs the Director to search for the oldest used Volume in the Pool when another Volume is requested by the Storage Daemon and none are available. The catalog is then **purged** irrespective of retention periods of all Files and Jobs written to this Volume. The Volume is then recycled and will be used as the next Volume to be written. This directive overrides any Job, File, or Volume retention periods that you may have specified.

This directive can be useful if you have a fixed number of Volumes in the Pool and you want to cycle through them and reusing the oldest one when all Volumes are full, but you don't want to worry about setting proper retention periods. However, by using this option you risk losing valuable data.

Be aware that **PurgeOldestVolume** disregards all retention periods. If you have only a single Volume defined and you turn this variable on, that Volume will always be immediately overwritten when it fills. So at a minimum, ensure that you have a decent number of Volumes in your Pool before running any jobs. If you want retention periods to apply do not use this directive. To specify a retention period, use the **VolumeRetention** directive (see above).

We highly recommend against using this directive, because it is sure that some day, **Bacula** will recycle a Volume that contains current data. The default is **no**.

File Retention = <time-period-specification> The File Retention directive defines the length of time that **Bacula** will keep File records in the Catalog database after the End time of the Job corresponding to the File records.

This directive takes precedence over Client directives of the same name. For example, you can decide to increase Retention times for Archive or OffSite Pool.

Note, this affects only records in the catalog database. It does not affect your archive backups.

For more information, see Client documentation about FileRetention

Job Retention = <time-period-specification> The Job Retention directive defines the length of time that **Bacula** will keep Job records in the Catalog database after the Job End time. As with the other retention periods, this affects only records in the catalog and not data in your archive backup.

This directive takes precedence over Client directives of the same name. For example, you can decide to increase Retention times for Archive or OffSite Pool.

For more information, see Client side documentation JobRetention

Cleaning Prefix = <string> This directive defines a prefix string, which if it matches the beginning of a Volume name during labeling of a Volume, the Volume will be defined with the VolStatus set to and thus **Bacula** will never attempt to use this tape. This is primarily for use with autochangers that accept barcodes where the convention is that barcodes beginning with **CLN** are treated as cleaning tapes.

Bacula supports ANSI or IBM tape labels as long as you enable it. In fact, with the proper configuration, you can force Bacula to require ANSI or IBM labels. Bacula can create an ANSI or IBM label, but if Check Labels is enabled

(see below), Bacula will look for an existing label, and if it is found, it will keep the label. Consequently, you can label the tapes with programs other than Bacula, and Bacula will recognize and support them. Even though Bacula will recognize and write ANSI and IBM labels, it always writes its own tape labels as well. When using ANSI or IBM tape labeling, you must restrict your Volume names to a maximum of six characters. If you have labeled your Volumes outside of Bacula, then the ANSI/IBM label will be recognized by Bacula only if you have created the HDR1 label with BACULA.DATA in the Filename field (starting with character 5). If Bacula writes the labels, it will use this information to recognize the tape as a Bacula tape. This allows ANSI/IBM labeled tapes to be used at sites with multiple machines and multiple backup pro- grams.

Label Type = ANSI | IBM | Bacula This directive is implemented in the Director Pool resource and in the SD Device resource. If it is specified in the SD Device resource, it will take precedence over the value passed from the Director to the SD. The default is Label Type = Bacula.

Label Format = <format> This directive specifies the format of the labels contained in this pool. The format directive is used as a sort of template to create new Volume names during automatic Volume labeling.

The should be specified in double quotes, and consists of letters, numbers and the special characters hyphen (**-**), underscore (_), colon (**:**), and period (**.**), which are the legal characters for a Volume name. The should be enclosed in double quotes (“).

In addition, the format may contain a number of variable expansion characters which will be expanded by a complex algorithm allowing you to create Volume names of many different formats. In all cases, the expansion process must resolve to the set of characters noted above that are legal Volume names. Generally, these variable expansion characters begin with a dollar sign (**$**) or a left bracket ([). If you specify variable expansion characters, you should always enclose the format with double quote characters (**”**). For more details on variable expansion, see the *Variable Expansion* chapter.

If no variable expansion characters are found in the string, the Volume name will be formed from the string appended with the a unique number that increases. If you do not remove volumes from the pool, this number should be the number of volumes plus one, but this is not guaranteed. The unique number will be edited as four digits with leading zeros. For example, with a **LabelFormat=”Vol-”**, the first volumes will be named Vol-0001, Vol-0002, etc.

With the exception of Job specific variables, you can test your **LabelFormat** by using the var command described in the Bacula Enterprise Console manual.

```
Label Format=":math:`{Level}_`\ Type\_\ :math:`{Client}_`\ Year-:math:`
 ↪{Month:p/2/0/r}-`\ Day:p/2/0/r"
```

Once defined, the name of the volume cannot be changed. When the volume is recycled, the volume can be used by an other Job at an other time, and possibly from an other Pool. In the example above, the volume defined with such name is probably not supposed to be recycled or reused.

In almost all cases, you should enclose the format specification (part after the equal sign) in double quotes.

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 bconsole** program. In addition to adding Volumes to the Pool (i.e. putting the Volume names in the Catalog database), the physical Volume must be labeled with a valid * *Bacula** software volume label before 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 Scratch Pool

In general, you can give your Pools any name you wish, but there is one important restriction: the Pool named **Scratch**, if it exists behaves like a scratch pool of Volumes in that when **Bacula** needs a new Volume for writing and it cannot find one, it will look in the Scratch pool, and if it finds an available Volume, it will move it out of the Scratch pool into the Pool currently being used by the job.

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*

Go to:

- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Catalog Resource

The Catalog Resource defines what catalog to use for the current job. Currently, **Bacula** can only handle a single database server (MySQL, PostgreSQL) that is defined when configuring **Bacula**. However, there may be as many Catalogs (databases) defined as you wish. For example, you may want each Client to have its own Catalog database, or you may want backup jobs to use one database and verify or restore jobs to use another database.

Since both MySQL and PostgreSQL are networked databases, they may reside either on the same machine as the Director or on a different machine on the network. See below for more details.

Catalog Start of the Catalog resource. At least one Catalog resource must be defined.

Name = <name> The name of the Catalog. No necessary relation to the database server name. This name will be specified in the Client resource directive indicating that all catalog data for that Client is maintained in this Catalog. This directive is required.

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

DB Name = <name> This specifies the name of the database. If you use multiple catalogs (databases), you specify which one here. If you are using an external database server rather than the internal one, you must specify a name that is known to the server (i.e. you explicitly created the **Bacula** tables using this name. This directive is required.

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

DB Socket = <socket-name> This is the name of a socket to use on the local host to connect to the database. This directive is used only by **MySQL**. Normally, if neither **DB Socket** or **DB Address** are specified, MySQL will use the default socket. If the DB Socket is specified, the server must reside on the same machine as the Director.

DB Address = <address> This is the host address of the database server. Normally, you would specify this instead of **DB Socket** if the database server is on another machine. In that case, you will also specify **DB Port**. This directive is used only by MySQL and PostgreSQL. This directive is optional.

DB Port = <port> This defines the port to be used in conjunction with **DBAddress** to access the database if it is on another machine. This directive is used only by **MySQL** and **PostgreSQL**. This directive is optional.

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

```
Catalog {
    Name = BaculaCatalog
    dbname = bacula;
    user = bacula;
    password = ""
}
```

or for a Catalog on another machine:

```
Catalog {
    Name = BaculaCatalog
    dbname = bacula
    user = bacula
    password = ""
    DB Address = remote.acme.com
    DB Port = 1234
}
```

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*

Go to:

- *Messages Resource*
- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Messages Resource

The Messages resource defines how messages are to be handled and destinations to which they should be sent.

Even though each daemon has a full message handler, within the File daemon and the Storage daemon, you will normally choose to send all the appropriate messages back to the Director. This permits all the messages associated with a single Job to be combined in the Director and sent as a single email message to the user, or logged together in a single file.

Each message that **Bacula** generates (i.e. that each daemon generates) has an associated type such as INFO, WARNING, ERROR, FATAL, etc. Using the message resource, you can specify which message types you wish to see and where they should be sent. In addition, a message may be sent to multiple destinations. For example, you may want all error messages both logged as well as sent to you in an email. By defining multiple messages resources, you can have different message handling for each type of Job (e.g. Full backups versus Incremental backups).

In general, messages are attached to a Job and are included in the Job report. There are some rare cases, where this is not possible, e.g. when no job is running, or if a communications error occurs between a daemon and the director. In those cases, the message may remain in the system, and should be flushed at the end of the next Job. However, since such messages are not attached to a Job, any that are mailed will be sent to **/usr/lib/sendmail**. On some systems, such as FreeBSD, if your sendmail is in a different place, you may want to link it to the the above location.

The records contained in a Messages resource consist of a **destination** specification followed by a list of **message-types** in the format:

destination = message-type1, message-type2, message-type3, ...

or for those destinations that need and address specification (e.g. email):

destination = address = message-type1, message-type2, message-type3, ... Where **destination** is one of a predefined set of keywords that define where the message is to be sent (**stdout**, **file**, ...), **message-type** is one of a predefined set of keywords that define the type of message generated by **Bacula** (ERROR, WARNING, FATAL, ...), and **address** varies according to the **destination** keyword, but is typically an email address or a filename.

The following are the list of the possible record definitions that can be used in a message resource.

Messages Start of the Messages records.

Name = <name> The name of the Messages resource. The name you specify here will be used to tie this Messages resource to a Job and/or to the daemon.

MailCommand = <command> In the absence of this resource, **Bacula** will send all mail using the following command:

```
mail -s "Bacula Message" <recipients>
```

In many cases, depending on your machine, this command may not work. However, by using the **MailCommand**, you can specify exactly how to send the mail. During the processing of the **command** part, normally specified as a quoted string, the following substitutions will be used:

```
%% = %
%b = Job Bytes
%c = Client's name
%C = If the job is a Cloned job (Only on director side)
%d = Daemon's name (Such as host-dir or host-fd)
%D = Director's name (Also valid on file daemon)
%e = Job Exit Status
```

(continues on next page)

(continued from previous page)

```
%E = Non-fatal Job Errors
%f = Job FileSet (Only on director side)
%F = Job Files
%h = Client address
%i = JobId
%I = Migration/Copy JobId (Only in Copy/Migrate Jobs)
%j = Unique Job id
%l = Job Level
%n = Job name
%o = Job Priority
%p = Pool name (Only on director side)
%P = Current PID process
%r = Recipients
%R = Read Bytes
%s = Since time
%S = Previous Job name (Only on file daemon side)
%t = Job type (Backup, ...)
%v = Volume name (Only on director side)
%w = Storage name (Only on director side)
%x = Spooling enabled? ("yes" or "no")
```

Note: Any **MailCommand** directive must be specified in the **Messages** resource **before** the desired **Mail**, **MailOnSuccess**, or **MailOnError** directive. In fact, each of those directives may be preceded by a different **MailCommand**.

The following is an example.

```
mailcommand = "/opt/bacula/bin/bsmtp -h mail.example.com -f \\"\(Bacula\) %r \\" -s \\
↪"Bacula: %t %e of %c %l\" %r\"
```

The **bsmtp** program is provided as part of **Bacula**. For additional details, please see the bsmtp – Customizing Your Email Messages section. Please test any **mailcommand** that you use to ensure that your bsmtp gateway accepts the addressing form that you use. Certain programs such as **Exim** can be very selective as to what forms are permitted particularly in the from part. Be careful, most of the samples use **%r** in the sender part of the **mailcommand**. This is a convenient way to setup the sender and the recipient at once. When you configure multiple recipients (separated by a comma) you must replace the **%r** in the sender part with only one valid email address.

OperatorCommand = <command> This resource specification is similar to the **MailCommand** except that it is used for Operator messages. The substitutions performed for the **MailCommand** are also done for this command. Normally, you will set this command to the same value as specified for the **MailCommand**. The **OperatorCommand** directive must appear in the **Messages** resource before the **Operator** directive.

<destination> = <message-type1>, <message-type2>, ... Where **destination** may be one of the following:

- **stdout** Send the message to standard output.
- **stderr** Send the message to standard error.
- **console** Send the message to the console (**Bacula** Console). These messages are held until the console program connects to the Director.

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

Where **address** depends on the **destination**.

The **destination** may be one of the following:

director Send the message to the Director whose name is given in the **address** field. Note, in the current implementation, the Director Name is ignored, and the message is sent to the Director that started the Job.

file Send the message to the filename given in the **address** field. If the file already exists, it will be overwritten.

append Append the message to the filename given in the **address** field. If the file already exists, it will be appended to. If the file does not exist, it will be created.

syslog Send the message to the system log (syslog) using the facility specified in the **address** field. The address field is ignored and the message is always sent to the LOG_DAEMON facility. The level of ERR, NOTICE, and INFO are used only.

See **man 3 syslog** for more details. Example:

```
syslog = all, !skipped
```

Although the **syslog** destination is not used in the default **Bacula** config files, in certain cases where **Bacula** encounters errors in trying to deliver a message, as a last resort, it will send it to the system **syslog** to prevent loss of the message, so you might occassionally check the **syslog** for **Bacula** output (normally **var/log/syslog**).

mail Send the message to the email addresses that are given as a comma separated list in the **address** field. Mail messages are grouped together during a job and then sent as a single email message when the job terminates. The advantage of this destination is that you are notified about every Job that runs. However, if you backup five or ten machines every night, the volume of email messages can be important. Some users use filter programs such as **procmail** to automatically file this email based on the Job termination code (see **mailcommand**).

mail on error Send the message to the email addresses that are given as a comma separated list in the **address** field if the Job terminates with an error condition. **MailOnError** messages are grouped together during a job and then sent as a single email message when the job terminates. This destination differs from the **mail** destination in that if the Job terminates normally, the message is totally discarded (for this destination). If the Job terminates in error, it is emailed. By using other destinations such as **append** you can ensure that even if the Job terminates normally, the output information is saved.

mail on success Send the message to the email addresses that are given as a comma separated list in the **address** field if the Job terminates normally (no error condition). **MailOnSuccess** messages are grouped together during a job and then sent as a single email message when the job terminates. This destination differs from the **mail** destination in that if the Job terminates abnormally, the message is totally discarded (for this destination). If the Job terminates normally, it is emailed.

operator Send the message to the email addresses that are specified as a comma separated list in the **address** field. This is similar to **mail** above, except that each message is sent as received. Thus there is one email per message. This is most useful for **mount** messages (see below).

console Send the message to the **Bacula** console.

catalog Send the message to the Catalog database. The message will be written to the table named **Log** and a timestamp field will also be added. This permits Job Reports and other messages to be recorded in the Catalog so that they can be accessed by reporting software. **Bacula** will prune the Log records associated with a Job when the Job records are pruned. Otherwise, **Bacula** never uses these records internally, so this destination is only used for special purpose programs (e.g. **Bweb**).

stdout Send the message to the standard output (normally not used).

stderr Send the message to the standard error output (normally not used).

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

events Event messages. ex: Daemon startup/shutdown, Console login/logout, commands... This class of message is not included in the **all** message type. Specific events can be specified in the form **events.string**. Each destination directive can support up to 10 sub events.

info General information messages.

warning Warning messages. Generally this is some unusual condition but not expected to be serious.

error Non-fatal error messages. The job continues running. Any error message should be investigated as it means that something went wrong.

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

terminate Message generated when the daemon shuts down.

notsaved Files not saved because of some error. Usually because the file cannot be accessed (i.e. it does not exist or is not mounted).

skipped Files that were skipped because of a user supplied option such as an incremental backup or a file that matches an exclusion pattern. This is not considered an error condition such as the files listed for the **notsaved** type because the configuration file explicitly requests these types of files to be skipped. For example, any unchanged file during an incremental backup, or any subdirectory if the no recursion option is specified.

mount Volume mount or intervention requests from the Storage daemon. These requests require a specific operator intervention for the job to continue.

restored The **ls** style listing generated for each file restored is sent to this message class.

saved The **ls** style listing generated for each file saved is sent to this message class.

all All message types except debug, events and saved.

security Security info/warning messages principally from unauthorized connection attempts.

alert Alert messages. These are messages generated by tape alerts.

volmgmt Volume management messages. Currently there are no volume management messages generated.

The following is an example of a valid Messages resource definition, where all messages except files explicitly skipped or daemon termination messages are sent by email to enforcement@sec.com. In addition all mount messages are sent to the operator (i.e. emailed to enforcement@sec.com). Finally all messages other than explicitly skipped files and files saved are sent to the console:

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

With the exception of the email address (changed to avoid junk mail from robot's), an example Director's Messages resource is as follows.

```
Messages {
   Name = Standard
   mailcommand = "bacula/bin/bsmtp -h mail.example.com -f \"\(Bacula\) %r\" -s \"Bacula:
↪%t %e of %c %l\" %r"
   operatorcommand = "bacula/bin/bsmtp -h mail.example.com -f \"\(Bacula\) %r\" -s \
↪"Bacula: Intervention needed for %j\" %r"
   MailOnError = security@example.com = all, !skipped, !terminate
   append = "bacula/working/log" = all, !skipped, !terminate
   operator = security@example.com = mount
    console = all, !skipped, !saved
}
```


The following is an example of a valid Messages resource definition where Bacula will keep track of the user activity. It is often called "Auditing".

```
Messages {
    Name = Standard
    console = all, !skipped, !saved
    append = bacula/working/bacula.log = all
    catalog = all, events
    append = bacula/working/audit.log = events, !events.bweb
}
```

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*

Go to:

- *Console Resource*
- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Console Resource

There are three different kinds of consoles, which the administrator or user can use to interact with the Director. These three kinds of consoles comprise three different security levels.

- The first console type is an **anonymous** or **default** console, which has full privileges. There is no console resource necessary for this type since the password is specified in the Director's resource and consequently such consoles do not have a name as defined on a **Name=** directive. Typically you would use it only for administrators.
- The second type of console is a "named console" defined within a Console resource in both the Director's configuration file and in the Console's configuration file. Both the names and the passwords in these two entries must match much as is the case for Client programs.

  This second type of console has absolutely no privileges except those explicitly specified in the Director's Console resource. Thus you can have multiple Consoles with different names and passwords, sort of like multiple users, each with different privileges.

Access privileges to commands and resources are provided by specifying access control lists in the Director's Console resource. These ACLs are specified by a directive followed by a list of access names. Examples of this are shown below.

- The third type of console is similar to the above mentioned one in that it requires a Console resource definition in both the Director and the Console. In addition, if the console name, provided on the directive, is the same as a Client name, that console is permitted to use the command to change the Address directive in the Director's client resource to the IP address of the Console. This permits portables or other machines using DHCP (non-fixed IP addresses) to "notify" the Director of their current IP address.

The Console resource is optional and need not be specified.

The following directives are permitted within the Director's configuration resource:

Name = <name> The name of the console. This name must match the name specified in the Console's configuration resource (much as is the case with Client definitions).

Password = <password> Specifies the password that must be supplied for a named **Bacula** Console to be authorized. The same password must appear in the resource of the Console configuration file. For better security, the password is never actually passed across the network but rather is used as a shared secret in a challenge response authentication scheme.

This directive is required.

If you have either **/dev/random** or **bc** on your machine, **Bacula** will generate a random password during the configuration process. When installing from reasonably built packages, the password should be generated during the installation. Otherwise it may be left blank or be set to some static text, in which case you should change it immediately after installation.

The password is plain text. It is not generated through any special process. However, it is preferable for security reasons to choose sufficiently long random text.

AuthenticationPlugin = <plugin-definition> Specifies a plugin to use by the authentication API framework which allows to configure a different set of authentication mechanisms (user credentials verification) using a dedicated Director plugin. This is called BPAM (Bacula Pluggable Authentication Modules). The first plugin available is a LDAP connector that is suitable to work with OpenLDAP or Active Directory. The new framework supports standard **user/password** and **Multi Factor Authentication** schemes which are fully driven by external plugins. On the client side, **bconsole** will perform the needed user interaction to collect required credentials. Bacula will still support all previous authentication schemas including **CRAM-MD5** and **TLS**. It is possible to configure **TLS Authentication** together with BPAM authentication raising the required security level. BPAM authentication is available for named Console resources only. To use this feature you have to load a dedicated Director plugin from the directory pointed to by the **Plugin Directory** Director resource configuration. This plugin should be listed in director `status` command output. Then you should configure the Console resource to use this plugin for authentication with the **Authentication Plugin** directive together with a named Console configuration in **bconsole.conf** as described in the *Console Configuration* chapter.

```
Console {
    Name = "ldapconsole"
    Password = "xxx"
    Authentication Plugin = "ldap:<parameters>"
}
```

where **parameters** are the space separated list of one or more plugin parameters:

- **url** - LDAP Directory service location, i.e. "url=ldap://10.0.0.1/"
- **binddn** - DN used to connect to LDAP Directory service to perform required **query**
- **bindpass** - DN password used to connect to LDAP Directory service to perform required **query**
- **query** - a query performed on LDAP Directory serice to find user for authentication. The query string is composed as **<basedn>/<filter>**. Where **<basedn>** is a DN search starting point and **<filter>** is a standard LDAP

search object filter which support dynamic string substitution: **%u** will be replaced by credential's username and **%p** by credential's password, i.e. `query=dc=bacula,dc=com/(cn=%u)`.

- **starttls** - instruct BPAM LDAP Plugin to use **StartTLS** extension if LDAP Directory service will support it and fallback to no TLS if this extension is not available.
- **starttlsforce** - does the same what **starttls** but resport error on fallback.

Some actually working configuration examples:

bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with OpenLDAP authentication example.

```
Console {
    Name = "bacula_ldap_console"
    Password = "xxx"
    Authentication Plugin = "ldap:url=ldap://ldapsrv/ binddn=cn=root,dc=bacula,dc=com␣
↪bindpass=secret query=dc=bacula,dc=com/(cn=%u) starttls"
    ...
}
```

bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with Active Directory authentication example.

```
Console {
    Name = "bacula_ad_console"
    Password = "xxx"
    Authentication Plugin = "ldap:url=ldaps://ldapsrv/ binddn=cn=bacula,ou=Users,
↪dc=bacula,dc=com bindpass=secret query=dc=bacula,dc=com/(&
↪(objectCategory=person)(objectClass=user)(sAMAccountName=%u))"
    ...
}
```

Note: TLS Directives in the Console resource of bacula-dir.conf

Bacula has built-in network encryption code to provide secure network transport. The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this functionality.

For more information how to enable TLS encryption, click here.

The following ACL related directives apply to named consoles as introduced above.

JobACL = <name-list> This directive is used to specify a list of Job resource names that can be accessed by the console. Without this directive, the console cannot access any of the Director's Job resources. Multiple Job resource names may be specified by separating them with commas, and/or by specifying multiple JobACL directives. For example, the directive may be specified as:

```
JobACL = kernsave, "Backup client 1", "Backup client 2"
JobACL = "RestoreFiles"
```

With the above specification, the console can access the Director's resources for the four jobs named on the JobACL directives, but for no others.

ClientACL = <name-list> This directive is used to specify a list of Client resources that can be accessed by the console.

RestoreClientACL = <name-list> This directive is used to specify a list of Client resource names that can be used by the console to run Restore Jobs on. The **ClientACL** is not affected by the **RestoreClientACL** directive.


```
ClientAcl = bacula-fd # backup and restore
RestoreClientAcl = test-fd # restore only
BackupClientAcl = production-fd # backup only
```

BackupClientACL = <name-list> This directive is used to specify a list of Client resource names that can be used by the console for backup jobs. The **ClientACL** is not affected by the **BackupClientACL** directive.

DirectoryACL = <name-list> This directive is used to specify a list of directories that can be accessed by a restore session. Without this directive, the console cannot restore any file. Multiple directories names may be specified by separating them with commas, and/or by specifying multiple **DirectoryACL** directives.

To be able to resture to the source location, the entry "/" should be in the list of allowed directories.

UserIdACL = <name-list> This directive is used to specify a list of user and group ids that can be accessed from a restore session. Without this directive, the console cannot restore any file. During a restore session, the Director will compute the restore list and will exclude files and directories that cannot be accessed. Bacula uses the LStat catalog field to retrieve **mode**, **uid** and **gid** information for each file and compares them with the UserIdACL elements. If a parent directory doesn't have a catalog entry, access to this directory will be automatically granted.

UID/GID names are resolved using the Unix function **getpwnam()** by the Director, whereas the stored values in the catalog will be numerical and unmodified from what is retrieved from the client at backup time. This implies that User UID/GID access control lists are only useful in environments with a common user id assignment.

Windows systems are not compatible with the **UserIdACL** feature. The use of **UserIdACL=*all*** is required to restore Windows systems from a restricted Console.

Multiple UID/GID names may be specified by separating them with commas, and/or by specifying multiple **UserIdACL** directives.

StorageACL = <name-list> This directive is used to specify a list of Storage resources that can be accessed by the console.

ScheduleACL = <name-list> This directive is used to specify a list of Schedule resources that can be accessed by the console.

PoolACL = <name-list> This directive is used to specify a list of Pool resource names that can be accessed by the console.

FileSetACL = <name-list> This directive is used to specify a list of names of FileSet resources that can be accessed by the console.

CatalogACL = <name-list> This directive is used to specify a list of Catalog resource names that can be accessed by the console.

Note: Multiple Catalogs are not recommended

Despite this functionality being available, we strongly recommend *against* using more than one catalog database. Data segregation and access control can be more robustly achieved using pools and possibly devices per separated entity, and setting proper ACLs for access to them.

CommandACL = <name-list> This directive is used to specify a list of console commands that can be executed by the console.

Note: Dot-commands and command completion

Features such as command completion in **bconsole** or drop-down list population in graphical user interfaces may require more commands to be enabled than expected at first glance. For example, **bconsole** uses the `.help` command to find completions for commands, and uses `.jobs`, `.clients` and others to provide completions for the respective resource

names. Thus, a **CommandACL** directive with a value list such as `CommandAcl = "run","status","restore", "list", "estimate",".jobs",".clients",".filesets", ".client", ".help"` may be useful.

WhereACL = <string> This directive permits you to specify where a restricted console can restore files to. If this directive is not specified, only the default restore location as defined in the restore job's configuration is permitted (often **/tmp/bacula-restores**). If ***all*** is specified any path the user enters will be accepted (which is not very secure), any other value specified (there may be multiple WhereACL directives) will restrict the user to use that path. For example, on a Unix system, if you specify "/", restores will be allowed to the original location.

Aside from Director resource names and console command names, the special keyword ***all*** can be specified in any of the above access control lists. When this keyword is present, any resource or command name (which ever is appropriate) will be accepted. An example configuration file can be found in the *Console Configuration* section.

Please note the ***all*** keyword must be used alone in an ACL list. The following definition will be interpreted as a "Glob Pattern" for any FileSet including the "all" string:

```
FileSetACL = "fs-test*", "*all*"
```

or

```
FileSetACL = "fs-test*"
FileSetACL = "*all*"
```

As of Bacula version 10.2, **ACL** Resource directives accept wildcard globbing such as ***.prod** or **job.client1.*** to ease the configuration.

```
ClientAcl = "test-*"
JobACL = "job-test-*"
FileSetACL = "fs-test*"
```

Console Multi-Factor Authentication Plugins

Time-based One-Time Password (TOTP) Authentication Plugin

The TOTP Authentication Plugin is compatible with the RFC 6238. Many smartphone apps are available to store the keys and compute the TOTP code.

```
Console {
   Name = "totpconsole"
   Password = "xxx"

   Authentication Plugin = "totp:<parameters>"
}
```

where `parameters` are the space separated list of one or more plugin parameters:

- **keydir** The directory where the TOTP keys are stored. `/opt/bacula/etc/conf.d/totp` by default.
- **no_qrcode** Do not generate QR code at the first connection. The Bacula administrator must provide the TOTP key to users manually using the `btotp` command for example.
- **sendcommand** Use the command specified by `sendcommand` to send the QRCode as a PNG image to the user.

The working configuration examples:

Listing 1: bacula-dir.conf - Console resource configuration for BPAM TOTP Plugin example.

```
# In bacula-dir.conf
Console {
  Name = "totpconsole"
  Password = "xxx"

  # New directive
  Authentication Plugin = "totp"
}

# in bconsole.conf
Console {
  Name = totpconsole
  Password = "xxx"              # Same as in bacula-dir.conf/Console
}
Director {
  Name = mydir-dir
  Address = localhost
  Password = notused
}
```

At the first console connection, if the TLS link is correctly setup (using the shared secret key), the plugin will generate a specific random key for the console and display a QR code in the console output. The user must then scan the QR code with his smartphone using an app such as Google Authenticator or Aegis (Opensource).

Note: The program `qrencode` (>=4.0) is used to convert the otpauth URL to a QR code. If the program is not installed the QR code can't be displayed.

The `btotp` command can be used to manipulate TOTP keys. For example, to generate the QR code in the terminal:

```
# /opt/bacula/bin/btotp -n totpconsole -q
```

To generate the otpauth URL in the terminal (can be used to generate a QRcode later) with `qrencode`:

```
# /opt/bacula/bin/btotp -n totpconsole -u otpauth://totp/Bacula:totpconsole?
↪secret=OR2TM22...&issuer=Bacula
```

In this example, the QR code is sent by email to the console owner:

```
# In bacula-dir.conf
Console {
   Name = "roberto"
   Password = "xxx"

   # New directive
   Authentication Plugin = "totp: sendcommand=\"mpack -s 'Bacula Console Access for %d' -
↪d /opt/bacula/etc/template %a  %c@acme.org\"\""
}

# in bconsole.conf
```

(continues on next page)

(continued from previous page)

```
Console {
  Name = roberto
  Password = "xxx"              # Same as in bacula-dir.conf/Console
}
Director {
  Name = mydir-dir
  Address = localhost
  Password = notused
}
```

At the first console connection, the plugin will generate a specific random key for the console and send a QR code PNG file via the command configured. In the example, the mail command mpack will send an email with the attachement to the email roberto@acme.org.

```
Subject: Bacula Console Access for mydirector-dir

Please find the QR Code needed to access the
Bacula director

---image attached---
```

The following variables can be used in the `sendcommand` string:

- **%a** - Path to the QR code png file
- **%d** - Director name
- **%c** - Console name

LDAP Authentication Plugin

A LDAP Authentication Plugin that is suitable to connect OpenLDAP or ActiveDirectory connector plugins is available.

```
Console {
   Name = "ldapconsole"
   Password = "xxx"

   Authentication Plugin = "ldap:<parameters>"
}
```

where `parameters` are the space separated list of one or more plugin parameters:

- **url** - LDAP Directory service location, i.e. “url=ldap://10.0.0.1/”
- **binddn** - DN used to connect to LDAP Directory service to perform required `query`
- **binidpass** - DN password used to connect to LDAP Directory service to perform required `query`
- **query** - a query performed on LDAP Directory serice to find user for authentication. The `query` string is composed as <basedn>/<filter>. Where ‘<basedn> is a DN search starting point and <filter> is a standard LDAP search object filter which support dynamic string substitution: `%u` will be replaced by credential’s username and `%p` by credential’s password, i.e. `query=dc=bacula,dc=com/(cn=%u)`.
- **starttls** - instruct BPAM LDAP Plugin to use StartTLS extension if LDAP Directory service will support it and follback to no TLS if this extension is not available.
- **starttlsforce** - does the same what ‘starttls‘ but resport error on fallback.

The working configuration examples:

Listing 2: bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with OpenLDAP authentication example.

```
Console {
  Name = "bacula_ldap_console"
  Password = "xxx"
  # New directive (on a single line)
  Authentication Plugin = "ldap:url=ldap://ldapsrv/    binddn=cn=root,dc=bacula,dc=com␣
↪bindpass=secret query=dc=bacula,dc=com(cn=%u) starttls"
  ...
 }
```

Listing 3: bacula-dir.conf - Console resource configuration for BPAM LDAP Plugin with Active Directory authentication example.

```
Console {
  Name = "bacula_ad_console"
  Password = "xxx"

  # New directive (on a single line)
  uthentication Plugin = "ldap:url=ldaps://ldapsrv/ binddn=cn=bacula,ou=Users,dc=bacula,
↪dc=com bindpass=secret query=dc=bacula,dc=com/(&
↪(objectCategory=person)(objectClass=user)(sAMAccountName=%u))"
  ...
 }
```

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*

Go to:

- *Counter Resource*
- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Counter Resource

The Counter Resource defines a counter variable that can be accessed by variable expansion used for creating Volume labels with the **LabelFormat** directive. See the LabelFormat directive in this chapter for more details.

Counter Start of the Counter resource. Counter directives are optional.

Name = <name> The name of the Counter. This is the name you will use in the variable expansion to reference the counter value.

Minimum = <integer> This specifies the minimum value that the counter can have. It also becomes the default. If not supplied, is **zero** assumed.

Maximum = <integer> This is the maximum value value that the counter can have. If not specified or set to zero, the counter can have a maximum value of 2,147,483,648 (2 to the 31 power). When the counter is incremented past this value, it is reset to the Minimum.

WrapCounter = <counter-name> If this value is specified, when the counter is incremented past the maximum and thus reset to the minimum, the counter specified on the **WrapCounter** is incremented. (This is not currently implemented).

Catalog = <catalog-name> If this directive is specified, the counter and its values will be saved in the specified catalog. If this directive is not present, the counter will be redefined each time that **Bacula** is started.

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*

Go to:

- *Statistics Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

Statistics Resource

The Resource defines the statistic collector function that can send information to a Graphite instance, to a CSV file or to **bconsole** with the **statistics** command (See for more information).

Statistics Start of the resource. Statistics directives are optional.

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

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

Interval = <time-interval> The **Intervall** directive instructs the Statistics thread how long it should sleep between every collection iteration. This directive is optional and the default value is **300** seconds.

Type = <CSV|Graphite> The **Type** directive specifies the Statistics backend, which may be one of the following: **CSV** or **Graphite**. This directive is required.

CSV is a simple file level backend which saves all required metrics with the following format to the file: **"<time>, <metric>, <value>\n"**

Where **<time>** is a standard Unix time (a number of seconds from 1/01/1970) with local timezone as returned by a system call **time**(), **<metric>** is a **Bacula** metric string and is a metric value which could be in numeric format (int/float) or a string **True** or **False** for boolean variable. The CSV backend requires the **File =** parameter.

Graphite is a network backend which will send all required metrics to a Graphite server. The Graphite backend requires the **Host=** and **Port=** directives to be set.

If the Graphite server is not available, the metrics are automatically spooled in the working directory. When the server can be reached again, spooled metrics are despooled automatically and the spooling function is suspended.

Metrics = <metricspec> The **Metrics** directive allow metric filtering and is a filter which enables to use ***** and **?** characters to match the required metric name in the same way as found in shell wildcard resolution. You can exclude filtered metric with **!** prefix. You can define any number of filters for a single Statistics. Metrics filter is executed in order as found in configuration. This directive is optional and if not used all available metrics will be saved by this statistics backend.

Example:

```
# Include all metric starting with "bacula.jobs"
Metrics = "bacula.jobs.*"

# Exclude any metric starting with "bacula.jobs"
Metrics = "!bacula.jobs.*"
```

Prefix = <string> The **Prefix** allows to alter the metrics name saved by statistics to distinguish between different installations or daemons. The prefix string will be added to metric name as: "<prefix>.<metric_name>" This directive is optional.

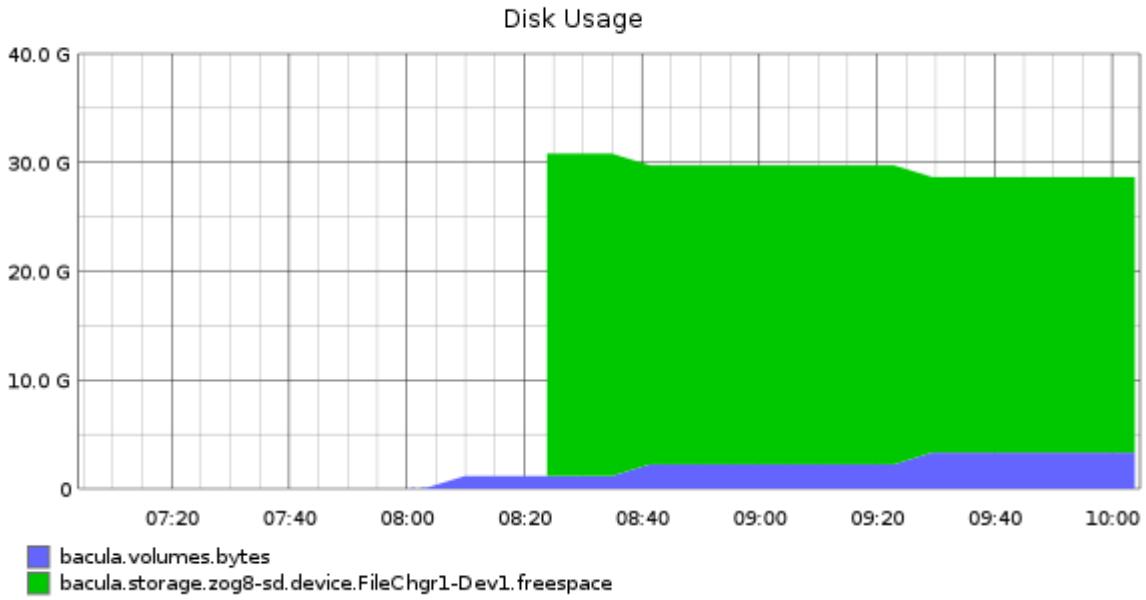
File = <filename> The File is used by the CSV statistics backend and point to the full path and filename of the file where metrics will be saved. With the CSV type, the **File** directive is required. The statistics thread must have the permissions to write to the selected file or create a new file if the file doesn't exist. If statistics is unable to write to the file or create a new one then the collection terminates and an error message will be generated. The file is only open during the dump and is closed otherwise. Statistics file rotation could be executed by a **mv** shell command.

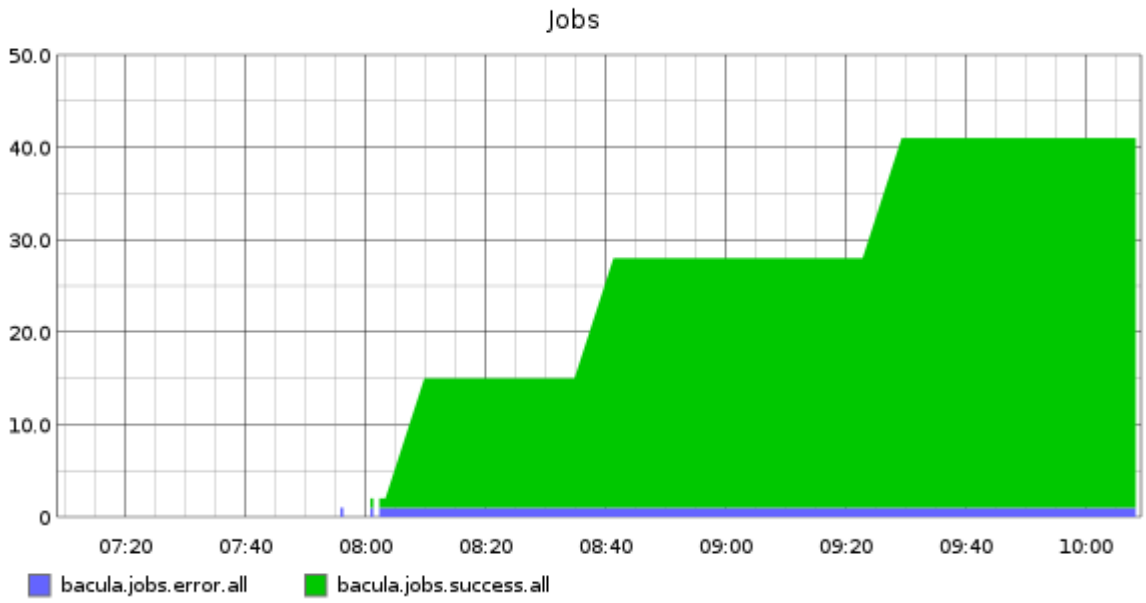
Host = <hostname> The **Host** directive is used for Graphite backend and specify the hostname or the IP address of the Graphite server. When the directive **Type** is set to Graphite, the Host directive is required.

Host = <number> The **Port** directive is used for Graphite backend and specify the TCP port number of the Graphite server. When the directive **Type** is set to Graphite, the **Port** directive is required.

Daemon Real-Time Statistics Monitoring

All daemons can now collect internal performance statistics periodically and provide mechanisms to store the values to a CSV file or to send the values to a Graphite daemon via the network. Graphite is an enterprise-ready monitoring tool (https://graphiteapp.org).





To activate the statistic collector feature, simply define a **Statistics** resource in the daemon of your choice:

```
Statistics {
    Name = "Graphite"
    Type = Graphite
    # Graphite host information
    Host = "localhost"
```

(continues on next page)

(continued from previous page)

```
    Port = 2003
}
```

It is possible to change the interval that is used to collect the statistics with the **Interval** directive (**5 mins** by default), and use the **Metrics** directive to select the data to collect (**all** by default).

If the Graphite daemon cannot be reached, the statistics data are spooled on disk and are sent automatically when the Graphite daemon is available again.

The **bconsole statistics** command can be used to display the current statistics in various formats (text or json for now).

```
*statistics
Statistics available for:
    1: Director
    2: Storage
    3: Client
Select daemon type for statistics (1-3): 1
bacula.dir.config.clients=1
bacula.dir.config.jobs=3
bacula.dir.config.filesets=2
bacula.dir.config.pools=3
bacula.dir.config.schedules=2
...
*statistics storage
...
bacula.storage.bac-sd.device.File1.readbytes=214
bacula.storage.bac-sd.device.File1.readtime=12
bacula.storage.bac-sd.device.File1.readspeed=0.000000
bacula.storage.bac-sd.device.File1.writespeed=0.000000
bacula.storage.bac-sd.device.File1.status=1
bacula.storage.bac-sd.device.File1.writebytes=83013529
bacula.storage.bac-sd.device.File1.writetime=20356
...
```

The **statistics bconsole** command can accept parameters to be scripted. For example, it is possible to export the data in JSON, or to select which metrics to display.

```
*statistics bacula.dir.config.clients bacula.dir.config.jobs json
[
    {
        "name": "bacula.dir.config.clients",
        "value": 1,
        "type": "Integer",
        "unit": "Clients",
        "description": "The number of defined clients in the Director."
    },
    {
    "name": "bacula.dir.config.jobs",
    "value": 3,
    "type": "Integer",
    "unit": "Jobs",
    "description": "The number of defined jobs in the Director."
    }
]
```


The **.status statistics** command can be used to query the status of the Statistic collector thread.

```
*.status dir statistics
Statistics backend: Graphite is running
    type=2 lasttimestamp=12-Sep-18 09:45
    interval=300 secs
    spooling=in progress
    lasterror=Could not connect to localhost:2003 Err=Connection refused
Update Statistics: running interval=300 secs lastupdate=12-Sep-18 09:45
*
```

See also:

Go back to:

- *Director Resource*
- *Job Resource*
- *JobDefs Resource*
- *Schedule Resource*
- *FileSet Resource*
- *Client Resource*
- *Storage Resource*
- *Autochanger Resource*
- *Pool Resource*
- *Catalog Resource*
- *Messages Resource*
- *Console Resource*
- *Counter Resource*

Go back to the *Director Resource Types* page.

Go back to the *Technical Reference for Director*.

See an example of a Director configuration file:

See also:

Go to:

- *Storage Daemon*
- *File Daemon*
- *Console*

Go back to the *Technical Reference for Director*.

1.2 Example Director Configuration File

An example Director configuration file might be the following:

```
#
# Default Bacula Director Configuration file
#
#  For Bacula release 16.0.6 (26 May 2023) -- redhat (Core)
#
#  You might want to change the default email address
#  from root to your email address.  See the "mail" and
#  "operator" directives in the Messages resource.
#
# Copyright (C) 2000-2020 Kern Sibbald
# License: BSD 2-Clause; see file LICENSE-FOSS
#

# This is the Director definition. Bacula consoles
# using the bconsole program will communicate with the
# Director using the Director name and the Password
# values defined in this resource.
# ------------------------------------------------------
Director {
  Name = bacula-dir
  DIRPort = 9101
  QueryFile = "/opt/bacula/scripts/query.sql"
  WorkingDirectory = "/opt/bacula/working"
  PIDDirectory = "/opt/bacula/working"
  MaximumConcurrentJobs = 20
  Password = "oHCFQnh1btsbM3ccDbQb7ULATlE/jGFA75Lh+FA9mbOv"
  Messages = Daemon
# CustomerId = TheCustomerId
}

# This is a JobDefs resource that can be used by any
# Job resources that share similar configurations. It
# is useful when multiple backup jobs differ, in just
# one or a few settings
# ------------------------------------------------------
JobDefs {
  Name = BackupsToDisk
  Type = Backup
  Client = bacula-fd
  Level = Incremental
  Schedule = Manual
  Storage = DiskAutochanger
  Messages = Default
  Pool = DiskBackup365d
  Priority = 10
  MaximumConcurrentJobs = 5
  Write Bootstrap = /opt/bacula/bsr/%c_%n.bsr
}
```

(continues on next page)

(continued from previous page)

```
# This Job resource uses the configuration defined in
# the BackupsToDisk JobDefs resource. This backup job
# will backup the home directory of the Client defined
# in the BackupsToDisk JobDefs resource, so no Client
# is required in this resource.
# --------------------------------------------------
Job {
  Name = LinuxHome
  JobDefs = BackupsToDisk
  FileSet = LinuxHome
}

# This Job resource uses the configuration defined in
# the BackupsToDisk JobDefs resource. This backup job
# will backup the etc directory of the Client defined
# in the BackupsToDisk JobDefs resource and it will use
# a different schedule.
# --------------------------------------------------
Job {
  Name = LinuxEtc
  JobDefs = BackupsToDisk
  FileSet = LinuxEtc
  Schedule = WeeklyFull
}

# This Job resource defines everything necessary to
# backup the Bacula configuration files. It does not
# make use of the the BackupsToDisk JobDefs.
# --------------------------------------------------
Job {
  Name = BaculaDirectorConfigs
  Type = Backup
  Client = bacula-fd
  Level = Incremental
  FileSet = BaculaConfigs
  Schedule = WeeklyFull
  Storage = DiskAutochanger
  Messages = Default
  Pool = DiskBackup365d
  Priority = 10
  Write Bootstrap = /opt/bacula/bsr/%c_%n.bsr
}

# This is the Catalog backup job. It should be run
# regularly when no other backup jobs are running to
# get a consistent backup of the Bacula catalog
# database.
# --------------------------------------------------
Job {
  Name = BaculaDirectorCatalog
  Type = Backup
  Level = Full
```

(continues on next page)

(continued from previous page)

```
  Client = bacula-fd
  FileSet= BaculaCatalog
  Schedule = DailyFull
  Storage = DiskCatalogStorage
  Messages = Default
  Pool = CatalogBackup14d
  Priority = 11
  WriteBootstrap = "/opt/bacula/bsr/%n.bsr"
  RunBeforeJob = "/opt/bacula/scripts/make_catalog_backup.pl BaculaCatalog"
  RunAfterJob  = "/opt/bacula/scripts/delete_catalog_backup"
  AllowDuplicateJobs = no
  CancelLowerLevelDuplicates = yes
  CancelQueuedDuplicates = yes
  AllowIncompleteJobs = no
}

# This is a Job of type "Restore". Many of the
# directives configured are place holders to satisfy
# the configuration parser and may be modified just
# prior to submitting the restore job.
# ------------------------------------------------------
Job {
  Name = Restore
  Type = Restore
  Client = bacula-fd
  FileSet= BaculaConfigs
  Storage = DiskAutochanger
  Pool = DiskBackup365d
  Messages = Default
  Where = "/opt/bacula/archive/bacula-restores"
  MaximumConcurrentJobs = 5
}

# This is a FileSet resource. It may be used by one or
# more backup jobs to backup the /opt/bacula/etc
# directory. An MD5 signature will be computed for all
# files saved.
# ------------------------------------------------------
FileSet {
  Name = BaculaConfigs
  Include {
    Options {
       Signature = md5
    }
    File = "/opt/bacula/etc"
  }
}

# This FileSet is used to backup the Bacula catalog
# dump file which is generated by a RunScript in the
# BaculaDirectorCatalog Job.
# ------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
FileSet {
  Name = BaculaCatalog
  Description = "This is used for the Catalog backup Job only"
  Include {
    Options {
      Signature = md5
    }
    File = "/opt/bacula/working/bacula.sql"
  }
}

# This FileSet may be used by one or more backup jobs
# to backup the /home directory.
# ------------------------------------------------------
FileSet {
  Name = LinuxHome
  Include {
    Options {
      Signature = md5
    }
    File = /home
  }
}

# This FileSet may be used by one or more backup jobs
# to backup the /etc directory.
# ------------------------------------------------------
FileSet {
  Name = LinuxEtc
  Include {
    Options {
      Signature = md5
    }
    File = /etc
  }
}

# List of files to be backed up
# ------------------------------------------------------
FileSet {
  Name = FullSet
  Include {
    Options {
      signature = MD5
    }
#
#  Put your list of files here, preceded by 'File =', one per line
#    or include an external list with:
#
#    File = <file-name
#
#  Note: / backs up everything on the root partition.
```

(continues on next page)

(continued from previous page)

```
#    if you have other partitions such as /usr or /home
#    you will probably want to add them too.
#
#  By default this is defined to point to the Bacula binary
#    directory to give a reasonable FileSet to backup to
#    disk storage during initial testing.
#
    File = /opt/bacula/bin
  }

#
# If you backup the root directory, the following two excluded
#   files can be useful
#
  Exclude {
    File = /opt/bacula/working
    File = /opt/bacula/archive
    File = /proc
    File = /tmp
    File = /sys
    File = /.journal
    File = /.fsck
  }
}

# This is a Schedule resource that may be used by any
# Job resource. Any Job that uses this schedule will
# run at Full level on Sundays at 23:05 and it will run
# at Incremental level from Mondays to Saturdays at
# 23:05.
# --------------------------------------------------------
Schedule {
  Name = WeeklyFull
  Run = Full Sun at 23:05
  Run = Incremental Mon-Sat at 23:05
}

# This is a Schedule resource for the Catalog job.
# It runs a daily Full backup at 2AM.
# --------------------------------------------------------
Schedule {
  Name = DailyFull
  Run = Full daily at 02:00
}

# This is a schedule that will never trigger any Jobs.
# It may be used in jobs just to clearly signify that
# the job is to be run manually.
# --------------------------------------------------------
Schedule {
  Name = Manual
}
```

(continues on next page)

(continued from previous page)

```
# This is a Client resource. Any File Daemon (Client)
# that this Director will contact must be configured in
# the Director configuration file. The Password value
# must be the same as the one defined in the Director
# resource in the File Daemon's bacula-fd.conf file.
# -------------------------------------------------------
Client {
  Name = bacula-fd
  Address = bacula
  FDPort = 9102
  Catalog = BaculaCatalog
  Password = Y5xRPsXwOHoZA7hbfzhUHzkTR9PcHKqDdfA3gmHQ+XUS
  FileRetention = 60 days
  JobRetention = 6 months
  AutoPrune = Yes
}

# This is an Autochanger resource. Any Autochanger
# defined in a Storage Daemon this Director will
# contact must be configured in the Director
# configuration file. The Password value must be the
# same as the one defined in the Director resource in
# the Storage Daemon's bacula-sd.conf file.
# -------------------------------------------------------
Autochanger {
  Name = DiskAutochanger
  Address = bacula
  SDPort = 9103
  Password = "Zc1a8NE61JbcWOlW1Pk8iLlJ39wxPo0EGNRMfTd6jHeh"
  Device = DiskAutochanger
  MediaType = DiskVolume
  MaximumConcurrentJobs = 10
}

# This is a Storage resource. It points to a single
# storage device in a Storage Daemon. Any Storage
# defined in a Storage Daemon that Director will
# contact muct be configured in the Director
# configuration file. The Password value must be the
# same as the one defined in the Director resource in
# the Storage Daemon's bacula-sd.conf file.
# -------------------------------------------------------
Storage {
  Name = DiskCatalogStorage
  Description = "Exclusively used for Catalog backups. Do not use for any other jobs!"
  Address = bacula
  SDPort = 9103
  Password = "Zc1a8NE61JbcWOlW1Pk8iLlJ39wxPo0EGNRMfTd6jHeh"
  Device = DiskCatalogDevice
  MediaType = CatalogVolume
  MaximumConcurrentJobs = 1
```

(continues on next page)

(continued from previous page)

```
}

# This is the Catalog resource. The connection
# settings to allow access to the Catalog database must
# be defined in this resource (database name, user,
# password, host, port, etc.).
# ------------------------------------------------------
Catalog {
  Name = BaculaCatalog
  DBname = "bacula"
  DBuser = "bacula"
  DBpassword = ""
}

# This is a Messages resource. It is used by jobs to
# report and log events.
# ------------------------------------------------------
Messages {
  Name = Default
  MailCommand = "/opt/bacula/bin/bsmtp -h localhost -f \"\(Bacula\) \<%r\>\" -s \
↪"Bacula: %t %e of %c %l\" %r"
  OperatorCommand = "/opt/bacula/bin/bsmtp -h localhost -f \"\(Bacula\) \<%r\>\" -s \
↪"Bacula: Intervention needed for %j\" %r"
  Mail = root@localhost = All, !Skipped
  Operator = root@localhost = Mount
  Console = All, !Skipped, !Saved
  Append = "/opt/bacula/log/bacula.log" = All, !Skipped
  Catalog = All
}

# This is another Messages resource. It is used by the
# Director daemon to report and log events.
# ------------------------------------------------------
Messages {
  Name = Daemon
  MailCommand = "/opt/bacula/bin/bsmtp -h localhost -f \"\(Bacula\) \<%r\>\" -s \"Bacula␣
↪daemon message\" %r"
  Mail = root@localhost = All, !Skipped
  Console = All
  Append = "/opt/bacula/log/bacula.log" = All, !Skipped
  Catalog = All
}

# This is a Pool resource. A pool represents a set of
# volumes. The volumes created in this example Pool
# will have a retention period of 365 days and they
# will have a size limit of 50G. This Pool allows up
# to 100 volumes. Volumes can be automatically recycled
# and reused (Recycle = Yes and AutoPrune = Yes).
# ------------------------------------------------------

Pool {
```

(continues on next page)

(continued from previous page)

```
  Name = DiskBackup365d
  PoolType = Backup
  Recycle = Yes
  AutoPrune = Yes
  VolumeRetention = 365 days
  MaximumVolumeBytes = 50G
  MaximumVolumes = 100
  VolumeUseDuration = 23h
  LabelFormat = Vol-
}

# This is another Pool resource which is used by the
# BaculaDirectorCatalog Job. Volumes in this Pool will
# have a retention period of 14 days, and a limit of
# one job per volume. Once the limit of 16 volumes is
# reached, volumes can be automatically recycled and
# reused (Recycle = Yes and AutoPrune = Yes).
# ----------------------------------------------------
Pool {
  Name = CatalogBackup14d
  PoolType = Backup
  Recycle = Yes
  AutoPrune = Yes
  VolumeRetention = 14 days
  MaximumVolumeJobs = 1
  MaximumVolumes = 16
  LabelFormat = CatalogVol-
}
# BEGIN ANSIBLE MANAGED BLOCK - include any file in the bee role files directory
@|"sh -c 'for f in /opt/bacula/etc/ansible.conf.d/director/*.conf ; do echo @${f} ; done'
↪"
# END ANSIBLE MANAGED BLOCK - include any file in the bee role files directory
```

See also:

Go back to *Director Resource Types*.

Go back to the *Technical Reference for Director*.

See also:

Go to:

- *Storage Daemon*
- *File Daemon*
- *Console*

Go back to the main *Technical Reference page*.

2 Storage Daemon

The following chapter aims at presenting the reader with Bacula technical content regarding the Storage Daemon.

The Storage Daemon configuration file has relatively few resource definitions. However, due to the great variation in backup media and system capabilities, the Storage Daemon must be highly configurable. As a consequence, there are quite a large number of directives in the *Device Resource* definition that allow you to define all the characteristics of your Storage device.

Read more:

2.1 Storage Daemon Resource Types

For a general discussion of configuration file and resources including the data types recognized by **Bacula**, 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 (disk or tape drive, for example).
- *Messages* – to define where to 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 directive is required.

Working Directory = <Directory> ** This directive is mandatory and specifies a directory in which the Storage daemon may put its status files. This directory should be used only by **Bacula, but may be shared by other **Bacula** daemons provided the names given to each daemon are unique. This directive is required.

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

Dedup Directory = <Directory> This directive is mandatory when using Global Endpoint Deduplication feature, and specifies a directory in which the Storage daemon may put data block files for all devices of the type **dedup**. The **Dedup Directory** can be very large after some point, we advise you to use a logical volume manager to be able to extend the filesystem when needed with new disks.

Dedup Index Directory = <Directory> This directive is optional and specifies a directory in which the Storage daemon may put its index files for all devices of the type **dedup**.

Encryption Command = <command> The **command** specifies an external program that must provide the encryption keys for the volumes. This is related to the **Volume Encryption** directive in **Device** resource. More information can be found in the Volume Encryption chapter.

Heartbeat Interval = <time-interval> This directive defines an interval of time in seconds. When the Storage daemon is waiting for the operator to mount a tape, each time interval, it will send a heartbeat signal to the File daemon. The default interval is **300s**. This feature is particularly useful if you have a router that does not follow Internet standards and times out a valid connection after a short duration despite the fact that keepalive is set. This usually results in a broken pipe error message.

Client Connect Wait = <time-interval> This directive defines an interval of time in seconds that the Storage daemon will wait for a Client (the File daemon) to connect. The default is **30 minutes**. Be aware that the longer the Storage daemon waits for a Client, the more resources will be tied up.

Maximum Concurrent Jobs = <number> where **<number>** is the maximum number of Jobs that may run concurrently. The default is set to **20**, but you may set it to a larger number. Each contact from the Director (e.g. status request, job start request) is considered as a Job, so if you want to be able to do a **status** request in the console at the same time as a Job is running, you will need to set this value greater than 1. To run simultaneous Jobs, you will need to set a number of other directives in the Director's configuration file. Which ones you set depend on what you want, but you will almost certainly need to set the **Maximum Concurrent Jobs** in the Storage resource in the Director's configuration file and possibly those in the Job and Client resources.

SDAddresses = <IP-address-specification> Specify the ports and addresses on which the Storage daemon will listen for Director connections. Normally, the default is sufficient and you do not need to specify this directive. Probably the simplest way to explain how this directive works is to show an example:

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

where ip, ip4, ip6, addr, and port are all keywords.

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

Using this directive, you can replace both the SDPort and SDAddress directives shown below.

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

SDAddress = <IP-Address> This directive is optional, and if it is specified, it will cause the Storage daemon server (for Director and File daemon connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this directive is not specified, the Storage Daemon will bind to any available address (the default).

FIPS Require = <yes|no> Require FIPS cryptographic module to start the daemon.

CommCompression = <yes|no> If the two **Bacula** components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes.

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is **enabled**. In the case that the compression is not effective, **Bacula** turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, **Bacula** reports **None** in the Job report.

Note: TLS Directives in the Storage resource of bacula-sd.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh** . The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

The following is a typical Storage daemon Storage definition.

```
#
# "Global" Storage daemon configuration specifications appear
# under the Storage resource.
#
Storage {
    Name = "bacula-sd"
    Address = baculasd.example.com
    WorkingDirectory = "/opt/bacula/working"
    PidDirectory = "/opt/bacula/working"
}
```

See also:

Go to:

- *Director Resource*
- *Device Resource*
- *Cloud Resource*
- *Autochanger Resource*
- *Messages Resource*
- *Statistics Resource*

Go back to *Storage Daemon Resource Types* page.

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

Director Resource

The Director resource specifies the Name of the Director which is permitted to use the services of the Storage daemon. There may be multiple Director resources. The Director Name and Password must match the corresponding values in the Director's configuration file.

Name = <Director-Name> Specifies the Name of the Director allowed to connect to the Storage daemon. This directive is required.

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

Monitor = <yes|no> If Monitor is set to **no** (default), this director will have full access to this Storage daemon. If Monitor is set to **yes**, this director will only be able to fetch the current status of this Storage daemon.

Note: If this director is being used by a Monitor, we highly recommend to set this directive to **yes** to avoid serious security problems.

Note: TLS Directives in the Director resource of bacula-dir.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh**. The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

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

```
Director {
    Name = bacula-dir
    Password = my_secret_password
}
```

See also:

Go back to:

- *Storage Resource*

Go to:

- *Device Resource*
- *Cloud Resource*
- *Autochanger Resource*
- *Messages Resource*
- *Statistics Resource*

Go back to *Storage Daemon Resource Types* page.

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

Device Resource

The Device Resource specifies the details of each device 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 system name of the device is specified on the **Archive Device** directive described below. The name you specify here is also used in your Director's conf file on the *Device directive* in its Storage resource.

Archive Device = <name-string> The specified **name-string** gives the system file name of the storage device managed by this storage daemon. This will usually be the device file name of a storage device. In the case of a tape device, it can be, for example **/dev/nst0** or **/dev/rmt/0mbn** . It may also be a directory name if you are archiving to disk storage. In this case, you must supply the full absolute path to the directory. When specifying a tape device, it is preferable that the "non-rewind" variant of the device file name be given. In addition, on systems such as Sun, which have multiple tape access methods, you must be sure to specify to use Berkeley I/O conventions with the device. The **b** in the Solaris (Sun) archive specification **/dev/rmt/0mbn** is what is needed in this case. **Bacula** does not support SysV tape drive behavior.

As noted above, normally the Archive Device is the name of a tape drive, but you may also specify an absolute path to an existing directory. If the Device is a directory **Bacula** will write to file storage in the specified directory, and the filename used will be the Volume name as specified in the Catalog. If you want to write into more than one directory (i.e. to spread the load to different disk drives), you will need to define two Device resources, each containing an Archive Device with a different directory.

In addition to a tape device name or a directory name, **Bacula** will accept the name of a FIFO. A FIFO is a special kind of file that connects two programs via kernel memory. If a FIFO device is specified for a backup operation, you must have a program that reads what **Bacula** writes into the FIFO. When the Storage Daemon starts the job, it will wait for **MaximumOpenWait** seconds for the read program to start reading, and then time it out and terminate the job. As a consequence, it is best to start the read program at the beginning of the job perhaps with the **RunBeforeJob** directive. For this kind of device, you never want to specify **AlwaysOpen**, because you want the Storage daemon to open it only when a job starts, so you must explicitly set it to **no**. Since a FIFO is a one way device, **Bacula** will not attempt to read a label of a FIFO device, but will simply write on it. To create a FIFO Volume in the catalog, use the **add** command rather than the **label** command to avoid attempting to write a label.

```
Device {
    Name = FifoStorage
    Media Type = Fifo
    Device Type = Fifo
    Archive Device = /tmp/fifo
    LabelMedia = yes
    Random Access = no
    AutomaticMount = no
    RemovableMedia = no
    MaximumOpenWait = 60
    AlwaysOpen = no
}
```

During a restore operation, if the Archive Device is a FIFO, **Bacula** will attempt to read from the FIFO, so you must have an external program that writes into the FIFO. **Bacula** will wait **MaximumOpenWait** seconds for the program to begin writing and will then time it out and terminate the job. As noted above, you may use the **RunBeforeJob** to start the writer program at the beginning of the job.

The Archive Device directive is required.

Device Type = <type-specification> The Device Type specification allows you to explicitly tell **Bacula** what kind of device you are defining. It the *type-specification* may be one of the following:

- **File** Tells **Bacula** that the device is a file. It may either be a file defined on fixed medium or a removable filesystem such as USB. All files must be random access devices.
- **Tape** The device is a tape device and thus is sequential access. Tape devices are controlled using ioctl() calls.
- **Fifo** The device is a first-in-first out sequential access read-only or write-only device.
- **Aligned** Tells **Bacula** that the device is special Aligned Device. See the specific Aligned User's guide for more information.
- **Dedup** The device is a Deduplication device. The Storage Daemon will use the deduplication engine. Please see the specific Deduplication User's guide for more information.
- **Cloud** The device is a Cloud device. The Storage Daemon will use the Cloud to upload/download volumes. Please see the specific Cloud User's guide for more information.

The Device Type directive is not required, and if not specified, **Bacula** will attempt to guess what kind of device has been specified using the Archive Device specification supplied. There are several advantages to explicitly specifying the Device Type. First, on some systems, block and character devices have the same type. Secondly, if you explicitly specify the Device Type, the mount point need not be defined until the device is opened. This is the case with most removable devices such as USB that are mounted by the HAL daemon. If the Device Type is not explicitly specified, then the mount point must exist when the Storage daemon starts.

Cloud = <name-string> A Device's new Cloud directive specifies a Cloud resource. As with other Bacula resource specifications, this directive specifies the name of the Cloud resource that this device will use.

Example:

```
Cloud = MyCloud
```

Enabled = <yes|no> This directive allows you to enable or disable the resource.

Media Type = <name-string> The specified **name-string** names the type of media supported by this device, for example, "LTO-8". Media type names are arbitrary in that you set them to anything you want, but they must be known to the volume database to keep track of which storage daemons can read which volumes. In general, each different storage type should have a unique Media Type associated with it. The same **name-string** must appear in the appropriate Storage resource definition in the Director's configuration file.

Even though the names you assign are arbitrary (i.e. you choose the name you want), you should take care in specifying them because the Media Type is used to determine which storage device **Bacula** will select during restore. Thus you should probably use the same Media Type specification for all drives where the Media can be freely interchanged. This is not generally an issue if you have a single Storage daemon, but it is with multiple Storage daemons, especially if they have incompatible media.

For example, if you specify a Media Type of "LTO-8" then during the restore, **Bacula** will be able to choose any Storage Daemon that handles "LTO-8". If you have an autochanger, you might want to name the Media Type in a way that is unique to the autochanger, unless you wish to possibly use the Volumes in other drives. You should also ensure to have unique Media Type names if the Media is not compatible between drives. This specification is required for all devices.

In addition, if you are using disk storage, each Device resource will generally have a different mount point or directory. In order for **Bacula** to select the correct Device resource, each one must have a unique Media Type.

Autochanger = <yes|no> If **yes**, this device belongs to an automatic tape changer, and you must specify an **Autochanger** resource that points to the **Device** resources. You must also specify a **Changer Device**. If the Autochanger directive is set to **no** (default), the volume must be manually changed. You should also have an identical directive to the Storage resource in the Director's configuration file so that when labeling tapes you are prompted for the slot.

Changer Device = <name-string> The specified **name-string** must be the **generic SCSI** device name of the autochanger that corresponds to the normal read/write **Archive Device** specified in the Device resource. This generic

SCSI device name should be specified if you have an autochanger or if you have a standard tape drive and want to use the **Alert Command** (see below). For example, on Linux systems, for an Archive Device name of **/dev/nst0**, you would specify **/dev/sg0** for the Changer Device name. Depending on your exact configuration, and the number of autochangers or the type of autochanger, what you specify here can vary. This directive is optional. See the Using Autochangers chapter for more details of using this and the following autochanger directives.

Changer Command = <name-string> The **name-string** specifies an external program to be called that will automatically change volumes as required by **Bacula**. Normally, this directive will be specified only in the **AutoChanger** resource, which is then used for all devices. However, you may also specify the different **Changer Command** in each Device resource. Most frequently, you will specify the **Bacula** supplied **mtx-changer** script as follows:

```
Changer Command = "/opt/bacula/scripts/mtx-changer"
```

and you will install the **mtx** program on your system. An example of this command is in the default **bacula-sd.conf** file. For more details on the substitution characters that may be specified to configure your autochanger, see the *Autochangers* chapter. For FreeBSD users, you might want to see one of the several **chio** scripts in **examples/autochangers**.

Bacula supports ANSI or IBM tape labels as long as you enable it. In fact, with the proper configuration, you can force Bacula to require ANSI or IBM labels. Bacula can create an ANSI or IBM label, but if Check Labels is enabled (see below), Bacula will look for an existing label, and if it is found, it will keep the label. Consequently, you can label the tapes with programs other than Bacula, and Bacula will recognize and support them. Even though Bacula will recognize and write ANSI and IBM labels, it always writes its own tape labels as well. When using ANSI or IBM tape labeling, you must restrict your Volume names to a maximum of six characters. If you have labeled your Volumes outside of Bacula, then the ANSI/IBM label will be recognized by Bacula only if you have created the HDR1 label with BACULA.DATA in the Filename field (starting with character 5). If Bacula writes the labels, it will use this information to recognize the tape as a Bacula tape. This allows ANSI/IBM labeled tapes to be used at sites with multiple machines and multiple backup pro- grams.

Label Type = ANSI | IBM | Bacula This directive is implemented in the Director Pool resource and in the SD Device resource. If it is specified in the the SD Device resource, it will take precedence over the value passed from the Director to the SD.

Check Label = <yes|No> If **yes**, Bacula Storage Daemon will check the label type when reading a tape volume. It can be used when the pool contains a mix of IBM/ANSI and Bacula volumes.

Control Device = <name-string> The control device is the SCSI control device that corresponds to the **Archive Device**. The correspondance can be done via the **lssci -g** command.

```
/opt/bacula# lsscsi -g
[1:0:0:0] tape HP Ultrium 4-SCSI H61W /dev/st0 /dev/sg0
[1:0:0:1] tape HP Ultrium 4-SCSI H61W /dev/st1 /dev/sg1
[1:0:0:2] mediumx HP MSL G3 Series E.00 - /dev/sg2
```

Alert Command = <name-string> The **name-string** specifies an external program to be called at the completion of each Job after the device is released. The purpose of this command is to check for Tape Alerts, which are present when something is wrong with your tape drive (at least for most modern tape drives). The same substitution characters that may be specified in the Changer Command may also be used in this string. For more information, see the *Autochangers* chapter.

The directive in the Device resource can call the **tapealert** script that is installed in the scripts directory. It is defined as follows:

```
Device {
    Name = ...
    Archive Device = /dev/nst0
    Alert Command = "/opt/bacula/scripts/tapealert %l"
    Control Device = /dev/sg1 # must be SCSI ctl for /dev/nst0
```

(continues on next page)

(continued from previous page)

```
    ...
}
```

Once the above mentioned two directives (**Alert Command** and **ControlDevice**) are in place in each of your resources, **Bacula** will check for tape alerts at two points:

- After the Drive is used and it becomes idle.
- After each read or write error on the drive.

At each of the above times, **Bacula** will call the new **tapealert** script, which uses the **tapeinfo** program. The **tapeinfo** utility is part of the **apt** `sg3-utils` and **rpm** `sg3_utils` packages. Then for each tape alert that **Bacula** finds for that drive, it will emit a Job message that is either `INFO`, `WARNING`, or `FATAL` depending on the designation in the Tape Alert published by the T10 Technical Committee on SCSI Storage Interfaces. For the specification, see: http://www.t10.org/ftp/t10/document.02/02-142r0.pdf

Worm Command = <name-string> The **name-string** specifies an external program to be called when loading a new volume. The purpose of this command is to check if the current tape is a `WORM` tape. The same substitution characters that may be specified in the **Changer Command** ay also be used in this string.

The directive in the resource can call the **isworm** script that is installed in the scripts directory. It is defined as follows:

```
Device {
    Name = ...
    Archive Device = /dev/nst0
    Worm Command = "/opt/bacula/scripts/isworm %l"
    Control Device = /dev/sg1 # must be SCSI ctl for /dev/nst0
    ...
}
```

Bacula will call the **isworm** script, which uses the **tapeinfo** and **sdparm** program.

Drive Index = <number> The **Drive Index** that you specify is passed to the **mtx-changer** script and is thus passed to the **mtx** program. By default, the Drive Index is **zero**, so if you have only one drive in your autochanger, everything will work normally. However, if you have multiple drives, you must specify multiple **Bacula** Device resources (one for each drive). The first Device should have the Drive Index set to 0, and the second Device Resource should contain a Drive Index set to 1, and so on. This will then permit you to use two or more drives in your autochanger. Using the **Autochanger** resource, **Bacula** will automatically ensure that only one drive at a time uses the autochanger script, so you no longer need locking scripts as in the past – the default script works for any number of drives.

Autoselect = <yes|no> If this directive is set to **yes** (default), and the Device belongs to an autochanger, then when the Autochanger is referenced by the Director, this device can automatically be selected. If this directive is set to **no**, then the Device can only be referenced by directly using the Device name in the Director. This is useful for reserving a drive for something special such as a high priority backup or restore operations.

Maximum Concurrent Jobs = <num> Maximum Concurrent Jobs is a directive that permits setting the maximum number of Jobs that can run concurrently on a specified Device. Using this directive, it is possible to have different Jobs using multiple drives, because when the Maximum Concurrent Jobs limit is reached, the Storage Daemon will start new Jobs on any other available compatible drive. This facilitates writing to multiple drives with multiple Jobs that all use the same Pool.

Maximum Changer Wait = <time> This directive specifies the maximum time in seconds for **Bacula** to wait for an autochanger to change the volume. If this time is exceeded, **Bacula** will invalidate the Volume slot number stored in the catalog and try again. If no additional changer volumes exist, **Bacula** will ask the operator to intervene. The default is **5 minutes**.

Maximum Rewind Wait = <time> This directive specifies the maximum time in seconds for **Bacula** to wait for a rewind before timing out. If this time is exceeded, **Bacula** will cancel the job. The default is **5 minutes**.

Maximum Open Wait = <time> This directive specifies the maximum time in seconds 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.

Always Open = <yes|no> If yes (default), Bacula will always keep the device open unless specifically unmounted by the Console program. This permits Bacula to ensure that the tape drive is always available, and properly positioned. If you set AlwaysOpen to no, Bacula will only open the drive when necessary, and at the end of the Job if no other Jobs are using the drive, it will be freed. The next time Bacula wants to append to a tape on a drive that was freed, Bacula will rewind the tape and position it to the end. To avoid unnecessary tape positioning and to minimize unnecessary operator intervention, it is highly recommended that Always Open = yes. This also ensures that the drive is available when Bacula needs it. If you have Always Open = yes (recommended) and you want to use the drive for something else, simply use the unmount command in the Console program to release the drive. However, don't forget to remount the drive with mount when the drive is available or the next Bacula job will block. For File storage, this directive is ignored. For a FIFO storage device, you must set this to no. Note that if you set this directive to no, Bacula will release the tape drive between each job, and thus the next job will rewind the tape and position it to the end of the data. This can be a very time consuming operation. In addition, with this directive set to no, certain multiple drive autochanger operations will fail. We strongly recommend to keep Always Open set to yes.

Volume Poll Interval = <time> If the time specified on this directive is non-zero, after asking the operator to mount a new volume **Bacula** will periodically poll (or read) the drive at the specified interval to see if a new volume has been mounted. If the time interval is zero, no polling will occur. The default value is **5 mins**. This directive can be useful if you want to avoid operator intervention via the console. Instead, the operator can simply remove the old volume and insert the requested one, and **Bacula** on the next poll will recognize the new tape and continue. Please be aware that if you set this interval too small, you may excessively wear your tape drive if the old tape remains in the drive, since **Bacula** will read it on each poll. This can be avoided by ejecting the tape using the **Offline On Unmount** and the **Close on Poll** directives. However, if you are using a Linux 2.6 kernel or other OSes such as FreeBSD or Solaris, the Offline On Unmount will leave the drive with no tape, and **Bacula** will not be able to properly open the drive and may fail the job.

Volume Encryption = <No|yes|strong> This directive allows you to enable the encryption for the given File device. The default is No.

The encryption can be of 3 different types:

- **no** This is the default, the device don't do any encryption.
- **yes** The device encrypts the data, but all information in the *volume label* are left in clear text.
- **strong** The device encrypts the data, and obfuscate any information in the *volume label* except the one that are needed for the management of the volume. The fields that are obfuscate are: *volume name*.

The encryption keys are given by the key manager program specified by the **Encryption Command** directive in the **Storage** resource. More information can be found in the Volume Encryption chapter.

Close on Poll = <yes|no> If **yes**, **Bacula** close the device (equivalent to an unmount except no mount is required) and reopen it at each poll. Normally this is not too useful unless you have the **Offline on Unmount** directive set, in which case the drive will be taken offline preventing wear on the tape during any future polling. Once the operator inserts a new tape, **Bacula** will recognize the drive on the next poll and automatically continue with the backup. Please see above more more details.

Sync On Close = <yes|No> If **yes**, **Bacula** will sync the device at the end of each job and when closing the device. Normally it might be useful if you store the data on network filesystems. The default value is **no**.

Set Volume Read Only = <yes|No>

Warning: Only for file-based devices.

If **yes**, this device sets the permission of the volume file for each volume when setting it's status to 'Full'. Permissions are set back when the volume is relabeled/truncated. The Access Time (atime) file attribute is updated with the maximum value between the Device `MinimumVolumeProtectionTime` and the Volume Retention.

When this directive is combined with the NetApp SnapLock, DataDomain or HPE StoreOnce immutable feature and the volume is Full, there is no other way than to wait for the `MinimumVolumeProtectionTime` or `VolumeRetention` to expire so that the volume can be relabeled/reused. The default is **no**.

Set Volume Immutable = <yes|No>

> **Warning:** Only for file-based devices.

This feature can only be used if Bacula is run as a systemd service because only then, with proper capabilities set for the daemon, it's allowed to manage files attributes of Bacula Volumes.

If **yes**, this device sets the 'Immutable' file attribute for each volume when setting it's status to 'Full'. Attribute is cleared when the volume is relabeled/truncated.

When this directive is set and the volume is Full, there is no other way than to wait for the `MinimumVolumeProtectionTime`' or `VolumeRetention` to expire so that the volume can be relabeled/reused. The default is **no**.

Set Volume Append Only = <yes|No>

> **Warning:** Only for file-based devices.

This feature can only be used if Bacula is run as a systemd service because only then, with proper capabilities set for the daemon, it's allowed to manage file attributes of Bacula Volumes.

If **yes**, this device sets the `Append_Only` file attribute for each volume that is created (either automatically or using the label command), at the moment of writing first backup job. Attribute is cleared when the volume is relabeled/truncated. The default is **no**.

Minimum Volume Protection Time = <time-interval>

> **Warning:** Only for file-based devices.

This feature can only be used if Bacula is run as a systemd service because only then, with proper capabilities set for the daemon, it's allowed to manage Volume Files' attributes.

This directive defines the minimum interval of time in which the Storage Daemon respects the 'Immutable' file attribute which is set when marking volume as Full. Bacula won't allow relabeling or reusing such a volume before this period expires. When applied, the Storage Daemon may use the `VolumeRetention` if it is more protective.

When this directive is set to 0, the drive device will not be able to truncate/relabel volumes at all, effectively keeping volumes all the time for further usage.

The default is **30 days**.

Removable Media = <yes|no> If **yes**, this device supports removable media (for example, tapes). If **no**, media cannot be removed (for example, an intermediate backup area on a hard disk). If **Removable media** is enabled on a File device (as opposed to a tape) the Storage daemon will assume that device may be something like a USB device that can be removed or a simply a removable hard disk. When attempting to open such a device, if the Volume is not found (for File devices, the Volume name is the same as the Filename), then the Storage daemon will search the entire device looking for likely Volume names, and for each one found, it will ask the Director if the Volume can be used. If so, the

Storage daemon will use the first such Volume found. Thus it acts somewhat like a tape drive – if the correct Volume is not found, it looks at what actually is found, and if it is an appendable Volume, it will use it.

If the removable medium is not automatically mounted (e.g. udev), then you might consider using additional Storage daemon device directives such as **Requires Mount**, **Mount Point**, **Mount Command**, and **Unmount Command**, all of which can be used in conjunction with **Removable Media**.

Random Access = <yes|no> If **yes**, the archive device is assumed to be a random access medium which supports the **lseek** (or **lseek64** if Largefile is enabled during configuration) facility. This should be set to **yes** for all file systems such as USB, and fixed files. It should be set to **no** for non-random access devices such as tapes and named pipes.

Requires Mount = <yes|no> When this directive is enabled, the Storage daemon will submit a **Mount Command** before attempting to open the device. You must set this directive to **yes** for removable file systems such as USB devices that are not automatically mounted by the operating system when plugged in or opened by **Bacula**. It should be set to **no** for all other devices such as tapes and fixed filesystems. It should also be set to **no** for any removable device that is automatically mounted by the operating system when opened (e.g. USB devices mounted by udev or hotplug). This directive indicates if the device requires to be mounted using the **Mount Command**. To be able to write devices need a mount, the following directives must also be defined: **Mount Point**, **Mount Command**, and **Unmount Command**.

Mount Point = <directory> Directory where the device can be mounted. This directive is used only for devices that have **Requires Mount** enabled such as USB file devices.

Mount Command = <name-string> This directive specifies the command that must be executed to mount devices such as many USB devices. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

See the *Edit Codes* section below for more details of the editing codes that can be used in this directive.

If you need to specify multiple commands, create a shell script.

Unmount Command = <name-string> This directive specifies the command that must be executed to unmount devices such as many USB devices. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Unmount Command = "/bin/umount
```

See the *Edit Codes* section below for more details of the editing codes that can be used in this directive.

If you need to specify multiple commands, create a shell script.

Block Checksum = <yes|no> You may turn off the Block Checksum (CRC32) code that **Bacula** uses when writing blocks to a Volume. Doing so can reduce the Storage daemon CPU usage slightly. It will also permit **Bacula** to read a Volume that has corrupted data.

The default is **yes**– i.e. the checksum is computed on write and checked on read.

We do not recommend to turn this off particularly on older tape drives or for disk Volumes where doing so may allow corrupted data to go undetected.

Minimum block size = <size-in-bytes> On most modern tape drives, you will not need or want to specify this directive, and if you do so, it will be to make **Bacula** use fixed block sizes. This statement applies only to non-random access devices (e.g. tape drives). Blocks written by the storage daemon to a non-random archive device will never be smaller than the given **size-in-bytes**. The Storage daemon will attempt to efficiently fill blocks with data received from active sessions but will, if necessary, add padding to a block to achieve the required minimum size.

To force the block size to be fixed, as is the case for some non-random access devices (tape drives), set the **Minimum block size** and the **Maximum block size** to the same value (zero included). The default is that both the minimum and maximum block size are **zero** and the default block size is **64,512 bytes**.

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


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

Please note that if you specify a fixed block size as shown above, the tape drive must either be in variable block size mode, or if it is in fixed block size mode, the block size (generally defined by **mt**) **must** be identical to the size specified in **Bacula** – otherwise when you attempt to re-read your Volumes, you will get an error.

If you want the block size to be variable but with a 64K minimum and **256K** maximum (and default as well), you would specify:

```
Minimum block size = 64K
Maximum blocksize = 256K
```

The Device Minimum and Maximum Block Size directives must correspond to the settings used to write volumes in order to read them back correctly during a restore

Maximum block size = <size-in-bytes> On most modern tape drives, you will not need to specify this directive. If you do so, it will most likely be to reduce shoe-shine and improve performance on more modern LTO drives. The Storage daemon will always attempt to write blocks of the specified **size-in-bytes** to the archive device. As a consequence, this statement specifies both the default block size and the maximum block size. The size written never exceeds the given **size-in-bytes**. If adding data to a block would cause it to exceed the given maximum size, the block will be written to the archive device, and the new data will begin a new block.

If no value is specified or zero is specified, the Storage daemon will use a default block size of **64,512 bytes** (126 * 512).

The maximum **size-in-bytes** possible is 4,000,000.

Hardware End of Medium = <yes|no> If **no**, the archive device is not required to support end of medium ioctl request, and the storage daemon will use the forward space file function to find the end of the recorded data. If **yes**, the archive device must support the `ioctl MTEOM` call, which will position the tape to the end of the recorded data. In addition, your SCSI driver must keep track of the file number on the tape and report it back correctly by the **MTIOCGET** ioctl. Note, some SCSI drivers will correctly forward space to the end of the recorded data, but they do not keep track of the file number. On Linux machines, the SCSI driver has a **fast-eod** option, which if set will cause the driver to lose track of the file number. You should ensure that this option is always turned off using the program.

Default setting for Hardware End of Medium is **yes**. This function is used before appending to a tape to ensure that no previously written data is lost. We recommend if you have a non-standard or unusual tape drive that you use the **btape** program to test your drive to see whether or not it supports this function. All modern tape drives support this feature.

Fast Forward Space File = <yes|no> If **no**, the archive device is not required to support keeping track of the file number (**MTIOCGET** ioctl) during forward space file. If **yes**, the archive device must support the `ioctl MTFSF` call, which virtually all drivers support, but in addition, your SCSI driver must keep track of the file number on the tape and report it back correctly by the **MTIOCGET** ioctl. Note, some SCSI drivers will correctly forward space, but they do not keep track of the file number or more seriously, they do not report end of medium.

Default setting for Fast Forward Space File is **yes**.

Use MTIOCGET = <yes|no> If **no**, the operating system is not required to support keeping track of the file number and reporting it in the (**MTIOCGET** ioctl). The default is **yes**. If you must set this to No, **Bacula** will do the proper file position determination, but it is very unfortunate because it means that tape movement is very inefficient. Fortunately, this operation system deficiency seems to be the case only on a few *BSD systems. Operating systems known to work correctly are Solaris, Linux and FreeBSD.

BSF at EOM = <yes|no> If **no**, the default, no special action is taken by **Bacula** with the End of Medium (end of tape) is reached because the tape will be positioned after the last EOF tape mark, and **Bacula** can append to the tape as desired. However, on some systems, such as FreeBSD, when **Bacula** reads the End of Medium (end of tape), the tape will be positioned after the second EOF tape mark (two successive EOF marks indicated End of Medium). If **Bacula** appends from that point, all the appended data will be lost. The solution for such systems is to specify **BSF at EOM**

which causes **Bacula** to backspace over the second EOF mark. Determination of whether or not you need this directive is done using the **test** command in the **btape** program.

TWO EOF = <yes|no> If **yes** , **Bacula** will write two end of file marks when terminating a tape – i.e. after the last job or at the end of the medium. If **no**, the default, **Bacula** will only write one end of file to terminate the tape.

Backward Space Record = <yes|no> If **yes**, the archive device supports the `MTBSR ioctl` to backspace records. If **no**, this call is not used and the device must be rewound and advanced forward to the desired position. Default is **yes** for non random-access devices. This function if enabled is used at the end of a Volume after writing the end of file and any ANSI/IBM labels to determine whether or not the last block was written correctly. If you turn this function off, the test will not be done. This causes no harm as the re-read process is precautionary rather than required.

Backward Space File = <yes|no> If **yes**, the archive device supports the **MTBSF** and **MTBSF ioctl** s to backspace over an end of file mark and to the start of a file. If **no**, these calls are not used and the device must be rewound and advanced forward to the desired position. Default is **yes** for non random-access devices.

Forward Space Record = <yes|no> If **yes** , the archive device must support the **MTFSR ioctl** to forward space over records. If **no**, data must be read in order to advance the position on the device. Default is **yes** for non random-access devices.

Forward Space File = <yes|no> If **yes**, the archive device must support the `MTFSF ioctl` to forward space by file marks. If **no**, data must be read to advance the position on the device. Default is **yes** for non random-access devices.

Offline On Unmount = <yes|no> The default for this directive is **no**. If **yes** the archive device must support the `MTOFFL ioctl` to rewind and take the volume offline. In this case, **Bacula** will issue the offline (eject) request before closing the device during the **unmount** command. If **no Bacula** will not attempt to offline the device before unmounting it. After an offline is issued, the cassette will be ejected thus **requiring operator intervention** to continue, and on some systems require an explicit load command to be issued (**mt -f /dev/xxx load**) before the system will recognize the tape. If you are using an autochanger, some devices require an offline to be issued prior to changing the volume. However, most devices do not and may get very confused.

If you are using a Linux 2.6 kernel or other OSes such as FreeBSD or Solaris, the Offline On Unmount will leave the drive with no tape, and **Bacula** will not be able to properly open the drive and may fail the job.

Maximum Volume Size = <size> No more than **size** bytes will be written onto a given volume on the archive device. This directive is used mainly in testing **Bacula** to simulate a small Volume. It can also be useful if you wish to limit the size of a File Volume to say less than 50GB of data. In some rare cases of really antiquated tape drives that do not properly indicate when the end of a tape is reached during writing (though I have read about such drives, I have never personally encountered one). Please note, this directive is deprecated (being phased out) in favor of the **Maximum Volume Bytes** defined in the Pool resource in the Director's configuration file.

Maximum File Size = <size> No more than **size** bytes will be written into a given logical file on the volume. Once this size is reached, an end of file mark is written on the volume and subsequent data are written into the next file. Breaking long sequences of data blocks with file marks permits quicker positioning to the start of a given stream of data and can improve recovery from read errors on the volume. The default is **one Gigabyte**. This directive creates `EOF` marks only on tape media. However, regardless of the medium type (tape, disk, USB ...) each time the Maximum File Size is exceeded, a record is put into the catalog database that permits seeking to that position on the medium for restore operations. If you set this to a small value (e.g. 1MB), you will generate lots of database records (JobMedia) and may significantly increase CPU/disk overhead.

If you are configuring an LTO-7 or LTO-8 tape, you probably will want to set the **Maximum File Size** to **5GB** or bigger to avoid making the drive stop to write an `EOF` mark.

Note, this directive does not limit the size of Volumes that **Bacula** will create regardless of whether they are tape or disk volumes. It changes only the number of EOF marks on a tape and the number of block positioning records (see below) that are generated. If you want to limit the size of all Volumes for a particular device, use the **Maximum Volume Size** directive (above), or use **Maximum Volume Bytes** the directive in the Director's Pool resource, which does the same thing but on a Pool (Volume) basis.

Maximum File Index = <size> Some data might include information about the actual position of a block in the data stream. This information is stored in the catalog inside the table. By default, one index record will be created every 100MB of data. The index permits quicker positioning to the start of a given block in the **Bacula** Volume and can improve the Single Item Restore feature. If you set this to a small value (e.g. 1MB), you will generate lots of database records (FileMedia) and may significantly increase CPU/disk overhead.

Maximum Part Size = <size> This directive allows one to specify the maximum size for each part file of a cloud volume. Smaller part sizes will reduce restore costs, but may require a small additional overhead to handle multiple parts. The maximum number of parts permitted in a Cloud Volume is **524,288**. The maximum size of any given part is approximately 17.5TB.

Block Positioning = <yes|no> This directive tells **Bacula** not to use block positioning when doing restores. Turning this directive off can cause **Bacula** to be **extremely** slow when restoring files. You might use this directive if you wrote your tapes with **Bacula** in variable block mode (the default), but your drive was in fixed block mode. The default is **yes**.

Maximum Network Buffer Size = <bytes> where *bytes* specifies the initial network buffer size to use with the File daemon. This size will be adjusted down if it is too large until it is accepted by the OS. Please use care in setting this value since if it is too large, it will be trimmed by 512 bytes until the OS is happy, which may require a large number of system calls. The default value is **32,768** bytes. The maximum value is **1,000,000** bytes.

The default size was chosen to be relatively large but not too big in the case that you are transmitting data over Internet. It is clear that on a high speed local network, you can increase this number and improve performance. For example, some users have found that if you use a value of 65,536 bytes they get five to ten times the throughput. Larger values for most users don't seem to improve performance. If you are interested in improving your backup speeds, this is definitely a place to experiment. You will probably also want to make the corresponding change in each of your File daemons conf files.

Maximum Spool Size = <bytes> ** where the bytes specify the maximum spool size for all jobs that are running. The default is **no limit.**

Maximum Job Spool Size = <bytes> where the bytes specify the maximum spool size for any one job that is running. The default is **no limit**.

Spool Directory = <directory> specifies the name of the directory to be used to store the spool files for this device. This directory is also used to store temporary part files when writing to a device that requires mount (USB). The default is to use the **working directory** .

Use LinTape = <yes|no> If **yes**, the archive device must use the `IBM Lintape` Kernel interface instead of the standard `ST` Linux Kernel module. The `Use MTIOCGET = No` should be used when using the Lintape module. The default is **no**, the use of the default `ST` Linux Kernel module is recommended.

Label Media = <yes|no> If **yes**, permits this device to automatically label blank media without an explicit operator command. It does so by using an internal algorithm as defined on the Label Format record in each Pool resource. If this is **no** as by default, **Bacula** will label tapes only by specific operator command (**label** in the Console) or when the tape has been recycled. The automatic labeling feature is most useful when writing to disk rather than tape volumes.

Automatic mount = <yes|no> If **yes** (the default), permits the daemon to examine the device to determine if it contains a **Bacula** labeled volume. This is done initially when the daemon is started, and then at the beginning of each job. This directive is particularly important if you have set **Always Open = no** because it permits **Bacula** to attempt to read the device before asking the system operator to mount a tape. However, please note that the tape must be mounted before the job begins.

Edit Codes for Mount and Unmount Directives

Before submitting the **Mount Command**, **Unmount Command**, **Write Part Command**, or **Free Space Command** directives to the operating system, **Bacula** performs character substitution of the following characters:

```
%% = %
%a = Archive device name
%e = erase (set if cannot mount and first part)
%n = part number
%m = mount point
%v = last part name (i.e. filename)
```

Devices that Require a Mount (USB)

The directives "Requires Mount", "Mount Point", "Mount Command", and "Unmount Command" apply to removable filesystems such as USB.

Requires Mount = <yes|no> You must set this directive to **yes** for removable devices such as USB unless they are automatically mounted, and to **no** for all other devices (tape/disk). This directive indicates if the device requires to be mounted to be read, and if it must be written in a special way. If it set, **Mount Point**, **Mount Command**, **Unmount Command** directives must also be defined.

Mount Point = <directory> Directory where the device can be mounted.

Mount Command = <name-string> Command that must be executed to mount the device. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Mount Command = "/bin/mount -t iso9660 -o ro %a %m"
```

For some media, you may need multiple commands. If so, it is recommended that you use a shell script instead of putting them all into the Mount Command. For example, instead of this:

```
Mount Command = "/usr/local/bin/mymount"
```

Where that script contains:

```
#!/bin/sh
ndasadmin enable -s 1 -o w
sleep 2
mount /dev/ndas-00323794-0p1 /backup
```

Similar consideration should be given to all other Command parameters.

Unmount Command = <name-string> Command that must be executed to unmount the device. Before the command is executed, %a is replaced with the Archive Device, and %m with the Mount Point.

Most frequently, you will define it as follows:

```
Unmount Command = "/bin/umount %m"
```

If you need to specify multiple commands, create a shell script.

See also:

Go back to:

- *Storage Resource*
- *Director Resource*

Go to:

- *Cloud Resource*
- *Autochanger Resource*
- *Messages Resource*
- *Statistics Resource*

Go back to *Storage Daemon Resource Types* page.

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

Cloud Resource

The Cloud Storage Driver must be installed in the **Bacula** Storage Daemon **Plugin Directory** to be used.

Name = <Device-Name> The name of the Cloud resource. This is the logical Cloud name, and may be any string up to 127 characters in length.

Description = <Text> The description is used for display purposes as is the case with all resource.

Driver = <DriverName> This defines which driver to use. It can be **S3**, **Amazon**, **Azure**, **Google**, **Oracle** or **Swift**. There is also a **File** driver, which is used mostly for testing.

Host Name = <Name> This directive specifies the hostname to be used in the `URL`. Each Cloud service provider has a different and unique hostname. The maximum size is 255 characters and may contain a `TCP` port specification. This directive is not used with the Azure driver.

Bucket Name = <Name> This directive specifies the bucket name that you wish to use on the Cloud service. This name is normally a unique name that you create on the cloud service that identifies where you want to place your Cloud Volumes. The maximum bucket name size is 255 characters.

Access Key = <String> The access key is your unique user identifier given to you by your cloud service provider.

Secret Key = <String> The secret key is the security key that was given to you by your cloud service provider. It is equivalent to a password.

Protocol = <HTTP | HTTPS> The protocol defines the communications protocol to use with the cloud service provider. The two protocols currently supported are: `HTTPS` and `HTTP`. The default is HTTPS.

Uri Style = <VirtualHost | Path> This directive specifies the `URI` style to use to communicate with the cloud service provider. The two **Uri Styles** currently supported are: **VirtualHost** and **Path**. The default is **VirtualHost**.

Truncate Cache = <Truncate-kw> This directive specifies when **Bacula** should automatically remove (truncate) the local cache parts. Local cache parts will only be removed if they have been uploaded to the cloud. The currently implemented values are:

- **No** Do not remove cache. With this option you must manually delete the cache parts with the **bconsole truncate cache** command, or do so with an **Admin** Job that runs the **truncate cache** command. This is the default.
- **AfterUpload** Each part will be removed just after it is uploaded. Note, if this option **[2]**_ is specified, all restores will require a download from the Cloud.
- **AtEndOfJob** With this option **[3]**_, at the end of the Job, every part that has been uploaded to the Cloud will be removed (truncated).

Upload = <Upload-kw> This directive specifies when local cache parts will be uploaded to the Cloud. The options are:

- **Manual / No** Do not upload Volume cache parts automatically. With this option you must manually upload the Volume cache parts with a **bconsole Upload** command, or do so with an **Admin** Job that runs an **Upload** command. If not specified, this is the default.
- **EachPart** With this option, each cache Volume part will be uploaded when it is complete i.e. when the next Volume part is created or at the end of the Job.
- **AtEndOfJob** With this option all cache Volume parts that have not been previously uploaded will be uploaded at the end of the Job.

Maximum Concurrent Uploads = <number> The default is **3**, but by using this directive, you may set it to any value you want.

Maximum Concurrent Downloads = <number> The default is **3**, but by using this directive, you may set it to any value you want.

Maximum Upload Bandwidth = <speed> The default is **unlimited**, but by using this directive, you may limit the upload bandwidth used globally by all devices referencing this Cloud resource.

Maximum Download Bandwidth = <speed> The default is **unlimited**, but by using this directive, you may limit the download bandwidth used globally by all devices referencing this Cloud resource.

Region = <String> The Cloud resource can be configured to use a specific endpoint within a region. This directive is required for AWS-V4 regions, ex: **Region=. "eu-central-1"**

An example of a Cloud Resource might be:

```
Cloud {
    Name = AmazonCloud
    Driver = "Amazon"
    HostName = "s3.amazonaws.com"
    BucketName = "BaculaVolumes"
    AccessKey = "BZIXAIS39DYNER5FZ"
    SecretKey = "beesheeg7iTe0Gaex7aedie4aWohfuewohGaa0"
    Protocol = HTTPS
    UriStyle = VirtualHost
    Truncate Cache = No
    Upload = EachPart
    Region = "us-east-1"
    MaximumUploadBandwidth = 5MB/s
}
```

Transfer Priority = <High | Medium | Low> When restoring directly a part from Glacier, this directive indicates the rehydration priority level. Values can be **High**, **Medium** or **Low**. Default is **High**. Those values match respectively **Expeditive**, **Standard** and **Bulk** transfers tiers within S3.

Transfer Retention = <time-period-specification> This directive indicates the number of days the restored part must remain in S3's Reduced Redundancy Storage (RRS), available for download into the cache. At the end of this period the part will be removed from S3; the object will remain in Glacier. The minimum value is 1 day. The default is 5 days.

BlobEndpoint this directive can be used to specify a custom URL for the Azure Cloud blob (used by the Azure Driver only).

EndpointSuffix use this directive to specify a custom URL postfix for Azure (used by the Azure Driver only). Ex: **EnbpointSuffix="core.chinacloudapi.cn"**

StorageClass (available with Bacula Enterprise 14.0.5 and later) this directive can be used to specify the storage-Class for all parts transferred to the cloud, independently of the destination bucket class. Values can be **S3Standard**,

S3StandardIA, **S3IntelligentTiering**, **S3OneZoneIA**, **S3GlacierInstantRetrieval**, **S3GlacierFlexibleRetrieval**, **S3GlacierDeepArchive**, **S3Rrs**. (used by the Amazon Driver only).

See also:

Go back to:

- *Storage Resource*
- *Director Resource*
- *Device Resource*

Go to:

- *Autochanger Resource*
- *Messages Resource*
- *Statistics Resource*

Go back to *Storage Daemon Resource Types* page.

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

Autochanger Resource

The Autochanger resource supports single or multiple drive autochangers by grouping one or more Device resources into one unit called an autochanger in **Bacula** (often referred to as a "tape library" by autochanger manufacturers).

If you have an Autochanger, and you want it to function correctly, you **must** have an Autochanger resource in your Storage conf file, and your Director's Storage directives that want to use an Autochanger **must** refer to the Autochanger resource name.

Name = <Autochanger-Name> Specifies the Name of the Autochanger. This name is used in the Director's Storage resource definition to refer to the autochanger. This directive is required.

Device = <Device-name1, device-name2, . . . > Specifies the names of the Device resource or resources that correspond to the autochanger device. If you have a multiple device autochanger, you must specify multiple Device names, each one referring to a separate Device resource that contains a Drive Index specification that corresponds to the drive number base zero. You may specify multiple device names on a single line separated by commas, and/or you may specify multiple Device directives. This directive is required.

Changer Device = <name-string> The specified **<name-string>** gives the system file name of the autochanger device name. If specified in this resource, the Changer Device name is not needed in the Device resource. If it is specified in the Device resource (see above), it will take precedence over one specified in the Autochanger resource.

Changer Command = <name-string> The **<name-string>** specifies an external program to be called that will automatically change volumes as required by **Bacula**. Most frequently, you will specify the **Bacula** supplied **mtx-changer** script as follows. If it is specified here, it does not need to be specified in the Device resource. If it is also specified in the Device resource, it will take precedence over the one specified in the Autochanger resource.

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

```
Autochanger {
    Name = "LTO-8-changer"
    Device = LTO-8-1, LTO-8-2, LTO-8-3
    Changer Device = /dev/sg0
    Changer Command = "/opt/bacula/scripts/mtx-changer %c %o %S %a %d"
}
```

(continues on next page)

(continued from previous page)

```
Device {
    Name = "LTO-8-1"
    Drive Index = 0
    Media Type = LTO-8
    Autochanger = yes
    Archive Device = /dev/nst0
    Label Media = no
    Automatic mount = yes
    ...
}
Device {
    Name = "LTO-8-2"
    Drive Index = 1
    Media Type = LTO-8
    Autochanger = yes
    Archive Device = /dev/nst1
    Label Media = no
    Automatic mount = yes
    ...
}
Device {
    Name = "LTO-8-3"
    Drive Index = 2
    Media Type = LTO-8
    Autochanger = yes
    Autoselect = no
    Archive Device = /dev/nst2
    Label Media = no
    Automatic mount = yes
    ...
}
```

It is important to include the **Autochanger = yes** directive in each Device definition that belongs to an Autochanger. A device definition should not belong to more than one Autochanger resource. Also, your Device directive in the Storage resource of the Director's conf file should have the Autochanger's resource name rather than a name of one of the Devices.

If you have a drive that physically belongs to an Autochanger but you don't want to have it automatically used when **Bacula** references the Autochanger for backups, for example, you want to reserve it for restores, you can add the directive:

```
Autoselect = no
```

to the Device resource for that drive. In that case, **Bacula** will not automatically select that drive when accessing the Autochanger. You can, still use the drive by referencing it by the Device name directly rather than the Autochanger name. An example of such a definition is shown above for the Device LTO-8-3, which will not be selected when the name LTO-8-changer is used in a Storage definition, but will be used if LTO-8-3 is used.

Specifying Slots When Labeling

If you add an **Autochanger = yes** record to the Storage resource in your Director's configuration file, the **Bacula** Console will automatically prompt you for the slot number when the Volume is in the changer when you **add** or **label** tapes for that Storage device. If your **mtx-changer** script is properly installed, **Bacula** will automatically load the correct tape during the **label** command.

You must also set **Autochanger = yes** in the Storage daemon's Device resource as we have described above in order for the autochanger to be used. See the Storage Resource in the Director's chapter and the Device Resource in the Storage daemon chapter for more details on these records.

Thus all stages of dealing with tapes can be totally automated. It is also possible to set or change the Slot using **update** the command in the Console and selecting **Volume Parameters** to update.

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

If your autochanger has barcode labels, you can label all the Volumes in your autochanger one after another by using the **label barcodes** command. For each tape in the changer containing a barcode, **Bacula** will mount the tape and then label it with the same name as the barcode. An appropriate Media record will also be created in the catalog. Any barcode that begins with the same characters as specified on the **"CleaningPrefix=xxx** command, will be treated as a cleaning tape, and will not be labeled.

For example with:

```
Pool {
    Name ...
    Cleaning Prefix = "CLN"
    }
```

any slot containing a barcode of `CLNxxxx` will be treated as a cleaning tape and will not be mounted.

Please note that Volumes must be pre-labeled to be automatically used in the autochanger during a backup. If you do not have a barcode reader, this is done manually (or via a script).

Changing Cartridges

If you wish to insert or remove cartridges in your autochanger or you manually run the **mtx** program, you must first tell **Bacula** to release the autochanger by doing:

```
unmount
```

Change cartridges and/or run **mtx**

```
mount
```

If you do not do the **unmount** before making such a change, **Bacula** will become completely confused about what is in the autochanger and may stop function because it expects to have exclusive use of the autochanger while it has the drive mounted.

Dealing with Multiple Magazines

If you have several magazines or if you insert or remove cartridges from a magazine, you should notify **Bacula** of this. By doing so, **Bacula** will as a preference, use Volumes that it knows to be in the autochanger before accessing Volumes that are not in the autochanger. This prevents unneeded operator intervention.

If your autochanger has barcodes (machine readable tape labels), the task of informing **Bacula** is simple. Every time, you change a magazine, or add or remove a cartridge from the magazine, simply do

```
unmount
```

Remove magazine

Insert new magazine

```
update slots
mount
```

in the Console program. This will cause **Bacula** to request the autochanger to return the current Volume names in the magazine. This will be done without actually accessing or reading the Volumes because the barcode reader does this during inventory when the autochanger is first turned on. **Bacula** will ensure that any Volumes that are currently marked as being in the magazine are marked as no longer in the magazine, and the new list of Volumes will be marked as being in the magazine. In addition, the Slot numbers of the Volumes will be corrected in **Bacula**'s catalog if they are incorrect (added or moved).

If you do not have a barcode reader on your autochanger, you have several alternatives.

1. You can manually set the Slot and InChanger flag using the **update volume** command in the Console (quite painful).
2. You can issue a

   ```
   update slots scan
   ```

   command that will cause **Bacula** to read the label on each of the cartridges in the magazine in turn and update the information (Slot, InChanger flag) in the catalog. This is quite effective but does take time to load each cartridge into the drive in turn and read the Volume label.
3. You can modify the script so that it simulates an autochanger with barcodes. See below for more details.

Simulating Barcodes in Autochanger

You can simulate barcodes in your autochanger by making the **mtx-changer** script return the same information that an autochanger with barcodes would do. This is done by commenting out the one and only line in the **list**) case, which is:

```
:math: cat ${TMPFILE} | grep " Storage Element [0-9]*:.*Full" | awk "{print \$3 \$4}" |␣
↪sed "s/Full *\(:VolumeTag=\)*//"
```

by putting a # in column one of that line, or by simply deleting it. Then in its place add a new line that prints the contents of a file. For example:

```
cat /etc/bacula/changer.volumes
```

Be sure to include a full path to the file, which can have any name. The contents of the file must be of the following format:


```
1:Volume1
2:Volume2
3:Volume3 ...
```

Where the 1, 2, 3 are the slot numbers and Volume1, Volume2, ... are the Volume names in those slots. You can have multiple files that represent the Volumes in different magazines, and when you change magazines, simply copy the contents of the correct file into your **/etc/bacula/changer.volumes** file. There is no need to stop and start **Bacula** when you change magazines, simply put the correct data in the file, then run **update slots** the command, and your autochanger will appear to **Bacula** to be an autochanger with barcodes. [blb:updateslots]

The Full Form of the Update Slots Command

If you change only one cartridge in the magazine, you may not want to scan all Volumes, so the **update slots** command (as well as the **update slots scan** command) has the additional form:

```
update slots=n1,n2,n3-n4, ...
```

where the keyword **scan** can be appended or not. The n1,n2, ... represent Slot numbers to be updated and the form n3-n4 represents a range of Slot numbers to be updated (e.g. 4-7 will update Slots 4,5,6, and 7).

This form is particularly useful if you want to do a scan (time expensive) and restrict the update to one or two slots.

For example, the command:

```
update slots=1,6 scan
```

will cause **Bacula** to load the Volume in Slot 1, read its Volume label and update the Catalog. It will do the same for the Volume in Slot 6. The command:

```
update slots=1-3,6
```

will read the barcoded Volume names for slots 1,2,3 and 6 and make the appropriate updates in the Catalog. If you don't have a barcode reader or have not modified the **mtx-changer** script as described above, the above command will not find any Volume names so will do nothing.

Testing Autochanger and Adapting mtx-changer script

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 **/opt/bacula/scripts/mtx-changer**):

Make sure Bacula is not running.

/opt/bacula/scripts/mtx-changer /dev/sg0 list 0 /dev/nst0 0 This command should print:

```
1:
2:
3:
...
```

or one number per line for each slot that is occupied in your changer, and the number should be terminated by a colon (**:**). If your changer has barcodes, the barcode will follow the colon. If an error message is printed, you must resolve the problem (e.g. try a different SCSI control device name if **/dev/sg0** is incorrect). For example, on FreeBSD systems, the autochanger SCSI control device is generally **/dev/pass2**.

/opt/bacula/scripts/mtx-changer /dev/sg0 listall 0 /dev/nst0 0 This command should print:

```
Drive content:      D:Drive num:F:Slot loaded:Volume Name
D:0:F:2:vol2        or D:Drive num:E
D:1:F:42:vol42
D:3:E

Slot content:
S:1:F:vol1          S:Slot num:F:Volume Name
S:2:E               or S:Slot num:E
S:3:F:vol4

Import/Export tray slots:
I:10:F:vol10        I:Slot num:F:Volume Name
I:11:E              or I:Slot num:E
I:12:F:vol40
```

/opt/bacula/scripts/mtx-changer /dev/sg0 transfer This command should transfer a volume from source (1) to destination (2)

/opt/bacula/scripts/mtx-changer /dev/sg0 slots This command should return the number of slots in your autochanger.

/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0 If a tape is loaded from slot 1, this should cause it to be unloaded.

/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0 Assuming you have a tape in slot 3, it will be loaded into drive (0).

/opt/bacula/scripts/mtx-changer /dev/sg0 loaded 0 /dev/nst0 0 It should print "3" Note, we have used an "illegal" slot number 0. In this case, it is simply ignored because the slot number is not used. However, it must be specified because the drive parameter at the end of the command is needed to select the correct drive.

/opt/bacula/scripts/mtx-changer/dev/sg0unload3 /dev/nst00 will unload the tape into slot 3.

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. The only remaining area of problems will be if your autoloader needs some time to get the tape loaded after issuing the command. After the **mtx-changer** script returns, **Bacula** will immediately rewind and read the tape. If **Bacula** gets rewind I/O errors after a tape change, you will probably need to insert a **sleep 20** after the **mtx** command, but be careful to exit the script with a zero status by adding **exit 0** after any additional commands you add to the script. This is because **Bacula** checks the return status of the script, which should be zero if all went well.

You can test whether or not you need a **sleep** by putting the following commands into a file and running it as a script:

```
#!/bin/sh
/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0
mt -f/dev/st0 rewind
mt -f /dev/st0 weof
```

If the above script runs, you probably have no timing problems. If it does not run, start by putting a **sleep 30** or possibly a **sleep 60** in the script just after the **mtx-changer** load command. If that works, then you should move the sleep into the actual **mtx-changer** script so that it will be effective when **Bacula** runs.

A second problem that comes up with a small number of autochangers is that they need to have the cartridge ejected before it can be removed. If this is the case, the **load 3** will never succeed regardless of how long you wait. If this seems to be your problem, you can insert an eject just after the unload so that the script looks like:


```
#!/bin/sh
/opt/bacula/scripts/mtx-changer /dev/sg0 unload 1 /dev/nst0 0
mt -f /dev/st0 offline
/opt/bacula/scripts/mtx-changer /dev/sg0 load 3 /dev/nst0 0
mt -f /dev/st0 rewind
mt -f /dev/st0 weof
```

Obviously, if you need the **offline** command, you should move it into the **mtx-changer** script ensuring that you save the status of the **mtx** command or always force an **exit 0** from the script, because **Bacula** checks the return status of the script.

As noted earlier, there are several scripts in **<bacula-source>/examples/devices** that implement the above features, so they may be a help to you in getting your script to work.

If **Bacula** complains "Rewind error on **/dev/nst0**. ERR=Input/output error." you most likely need more sleep time in your **mtx-changer** before returning to **Bacula** after a load command has been completed.

Using the Autochanger

Let's assume that you have properly defined the necessary Storage daemon Device records, and you have added the **Autochanger = yes** record to the Storage 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:

```
/opt/bacula/bin/bconsole
Connecting to Director bacula-dir:9101
1000 OK: 10002 bacula-dir Version: 16.0.5 (05 April 2023)
Enter a period to cancel a command.
*label
```

it will then print something like:

```
Using Catalog "BaculaCatalog"
The defined Storage resources are:
        1: TapeAutochanger
        2: DiskAutochanger
Select Storage resource (1-2):1
```

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

```
Connecting to Storage daemon DiskAutochanger at baculasd:9103 ...
Enter autochanger drive[0]:
Enter new Volume name: LTO-8-Tape1
Enter slot (0 or Enter for none): 1
```

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

```
Defined Pools:
    1: Tape-pool
```

(continues on next page)

(continued from previous page)

```
    2: DiskBackup365d
Select the Pool (1-2): 1
```

I select the Tape-pool 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 TapeAutochanger at baculasd: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 Catalog "BaculaCatalog"
    Defined Pools:
        1: Tape-pool
        2: DiskBackup365d
    Select the Pool (1-2): 1
    +-------+-------------+--------+---------+-------+--------+----------+-------+------+
    | MedId | VolName     | MedTyp | VolStat | Bites | LstWrt | VolReten | Recyc | Slot |
    +-------+-------------+--------+---------+-------+--------+----------+-------+------+
    | 1     | LTO-8-Tape1 | LTO-8  | Append  | 0     | 0      | 30672000 | 0     | 1    |
    | 2     | LTO-8-Tape1 | LTO-8  | Append  | 0     | 0      | 30672000 | 0     | 2    |
    | 3     | LTO-8-Tape1 | LTO-8  | Append  | 0     | 0      | 30672000 | 0     | 3    |
    | ...                                                                            |
    +-------+-------------+--------+---------+-------+--------+----------+-------+------+
```

Barcode Support

Bacula provides barcode support with two Console commands, **label barcodes** and **update slots**.

The **label barcodes** will cause **Bacula** to read the barcodes of all the cassettes that are currently installed in the magazine (cassette holder) using the **mtx-changer list** command. Each cassette is mounted in turn and labeled with the same Volume name as the barcode.

The **update slots** command will first obtain the list of cassettes and their barcodes from **mtx-changer**. Then it will find each volume in turn in the catalog database corresponding to the barcodes and set its slot to correspond to the value just read. If the Volume is not in the catalog, then nothing will be done. This command is useful for synchronizing **Bacula** with the current magazine in case you have changed magazines or in case you have moved cassettes from one slot to another. If the autochanger is empty, nothing will be done.

The **Cleaning Prefix** statement can be used in the Pool resource to define a Volume name prefix, which if it matches that of the Volume (barcode) will cause that Volume to be marked with a VolStatus of **Cleaning**. This will prevent **Bacula** from attempting to write on the Volume.

Use bconsole to Display Autochanger Content

The **status slots storage=xxx** command displays autochanger content.

```
 Slot |  Volume Name    |  Status  |      Type          |    Pool         |  Loaded |
------+-----------------+----------+--------------------+-----------------+---------|
    1 |           00001 |   Append |  DiskChangerMedia  |         Default |    0    |
    2 |           00002 |   Append |  DiskChangerMedia  |         Default |    0    |
    3*|           00003 |   Append |  DiskChangerMedia  |         Scratch |    0    |
    4 |                 |          |                    |                 |    0    |
```

If you see a "*" near the slot number, you have to run **update slots** command to synchronize autochanger content with your catalog.

Setting Up a Tape Autochanger Using Bweb

Setting Up a Tape Autochanger Using Bweb

Bacula Autochanger Interface

Bacula calls the autochanger script that you specify on the **Changer Command** statement. Normally this script will be the **mtx-changer** script that we provide, but it can in fact be any program. The only requirement for the script is that it must understand the commands that **Bacula** uses, which are **loaded**, **load**, **unload**, **list**, and **slots**. In addition, each of those commands must return the information in the precise format as specified below:

Currently the changer commands used are:

loaded returns number of the slot that is loaded, base 1, in the drive or 0 if the drive is empty.

load loads a specified slot (note, some autochangers require a 30 second pause after this command) into the drive.

upload unloads the device (returns cassette to its slot).

list returns one line for each cassette in the autochanger in the format <slot>:<barcode>. Where the is the non-zero integer representing the slot number, and is the barcode associated with the cassette if it exists and if you autoloader supports barcodes. Otherwise the barcode field is blank.

slot returns total number of slots in the autochanger.

Bacula checks the exit status of the program called, and if it is zero, the data is accepted. If the exit status is non-zero, **Bacula** will print an error message and request the tape be manually mounted on the drive.

Autochanger Support

Bacula provides autochanger support for reading and writing tapes. In order to work with an autochanger, **Bacula** requires a number of things, each of which is explained in more detail after this list:

- A script that actually controls the autochanger according to commands sent by **Bacula**. We furnish such a script that works with found **mtx** in most linux distributions.
- That each Volume (tape) to be used must be defined in the Catalog and have a Slot number assigned to it so that **Bacula** knows where the Volume is in the autochanger. This is generally done with the **label** command, but can also done after the tape is labeled using the **update slots** command. See below for more details. You must pre-label the tapes manually before using them.
- Modifications to your Storage daemon's Device configuration resource to identify that the device is a changer, as well as a few other parameters.
- You should also modify your Storage resource definition in the Director's configuration file so that you are automatically prompted for the Slot when labeling a Volume.
- You need to ensure that your Storage daemon (if not running as root) has access permissions to both the tape drive and the control device.
- You need to have **Autochanger = yes** in your Storage resource in your **bacula-dir.conf** file so that you will be prompted for the slot number when you label Volumes.

The Autochanger resource permits you to group Device resources thus creating a multi-drive autochanger. If you have a tape library with multiple drives, you **must** use this resource.

Bacula uses its own **mtx-changer** script to interface with a program that actually does the tape changing. Thus in principle, **mtx-changer** can be adapted to function with any autochanger program, or you can call any other script or program. The current version of **mtx-changer** works with the **mtx** program. FreeBSD users have provided a script in the **examples/autochangers** directory that allows **Bacula** to use the **chio** program.

Bacula also supports autochangers with barcode readers. This support includes two Console commands: **label barcodes** and **update slots** . For more details on these commands, see the "Barcode Support" section below.

Current **Bacula** autochanger support does not include cleaning, stackers, or silos. Stackers and silos are not supported because **Bacula** expects to be able to access the slots randomly. However, if you are very careful to setup **Bacula** to access the Volumes in the autochanger sequentially, you may be able to make **Bacula** work with stackers (gravity feed and such).

In principle, **mtx** if will operate your changer correctly, then it is just a question of adapting the **mtx-changer** script (or selecting one already adapted) for proper interfacing.

If you are having troubles, please use the **auto** command in the **btape** program to test the functioning of your autochanger with **Bacula**. When **Bacula** is running, please remember that for many distributions (e.g. FreeBSD, Debian, ...) the Storage daemon runs as **bacula.tape** rather than **root.root**, so you will need to ensure that the Storage daemon has sufficient permissions to access the autochanger.

Some users have reported that the the Storage daemon blocks under certain circumstances in trying to mount a volume on a drive that has a different volume loaded. As best we can determine, this is simply a matter of waiting a bit. The drive was previously in use writing a Volume, and sometimes the drive will remain `BLOCKED` for a good deal of time (up to 7 minutes on a slow drive) waiting for the cassette to rewind and to unload before the drive can be used with a different Volume.

Knowing What SCSI Devices You Have

Under Linux, you can

```
cat /proc/scsi/scsi
```

to see what SCSI devices you have available. You can also:

```
cat /proc/scsi/sg/device_hdr /proc/scsi/sg/devices
```

to find out how to specify their control address (**/dev/sg0** for the first, **/dev/sg1** for the second, ...) on the **Changer Device = Bacula** directive.

You can also use the excellent **lsscsi** tool.

```
$ lsscsi -g
[1:0:2:0]     tape    SEAGATE    ULTRIUM06242-XXX    1619    /dev/st0    /dev/sg9
[1:0:14:0]    mediumx     STK         L180           0315    /dev/sch0   /dev/sg10
[2:0:3:0]     tape         HP    Ultrium 3-SCSI      G24S    /dev/st1    /dev/sg11
[3:0:0:0]     enclosu      HP        A6255A          HP04        -       /dev/sg3
[3:0:1:0]     disk     HP 36.4G    ST336753FC        HP00    /dev/sdd    /dev/sg4
```

For more detailed information on what SCSI devices you have please see the Linux SCSI Tricks problems section of the of the Tape Testing section of the Bacula Enterprise Problems Resolution Guide.

Under FreeBSD, you can use:

```
camcontrol devlist
```

To list the SCSI devices as well as the **/dev/passn** that you will use on the **Bacula Changer Device** directive.

Please check that your Storage daemon has permission to access this device.

The tip for FreeBSD users: on reboot **Bacula** will NOT have permission to control the device **/dev/pass0** (assuming this is your changer device). To get around this just edit the **/etc/devfs.conf** file and add the following to the bottom:

```
own pass0 root:bacula
perm pass0 0666
own nsa0.0 root:bacula
perm nsa0.0 0666
```

This gives the bacula group permission to write to the **nsa0.0** device too just to be on the safe side. To bring these changes into effect just run:

```
/etc/rc.d/devfs restart
```

Basically this will stop you having to manually change permissions on these devices to make **Bacula** work when operating the AutoChanger after a reboot.

The mtx-changer script

Please read the sections below so that you understand how autochangers work with **Bacula**. Although we supply a default **mtx-changer** script, your autochanger may require some additional changes.

Slots

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.

Bacula will not automatically use a Volume in your autochanger unless it is labeled and the slot number is stored in the catalog and the Volume is marked as InChanger. This is because it must know where each volume is (slot) to be able to load the volume.

For each Volume in your changer, you will, using the Console program, assign a slot. This information is kept in **Bacula's** catalog 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. When doing a **mount** command on an autochanger, you must specify which slot you want mounted. If the drive is loaded with a tape from another slot, it will unload it and load the correct tape, but normally, no tape will be loaded because an **unmount** command causes **Bacula** to unload the tape in the drive.

You can check if the Slot number and InChanger flag are set by doing a:

```
list Volumes
```

in the Console program.

Multiple Devices

Some autochangers have more than one read/write device (drive). The Autochanger resource permits you to group Device resources, where each device represents a drive. The Director may still reference the Devices (drives) directly, but doing so, bypasses the proper functioning of the drives together. Instead, the Director (in the Storage resource) should reference the Autochanger resource name. Doing so permits the Storage daemon to ensure that only one drive uses the **mtx-changer** script at a time, and also that two drives don't reference the same Volume.

Multi-drive requires the use of the **Drive Index** directive in the resource of the Storage daemon's configuration file. Drive numbers or the Device Index are numbered beginning at zero (0), which is the default. To use the second Drive in an autochanger, you need to define a second resource and set the ****Drive Index**** to 1 for that device. In general, the second device will have the same **Changer Device** (control channel) as the first drive, but a different **Archive Device**.

As a default, **Bacula** jobs will prefer to write to a Volume that is already mounted and available for writing. If you have a multiple drive autochanger and you want **Bacula** to write to more than one Volume in the same Pool at the same time, you will probably set MaximumConcurrentJobs to 1 in each tape device. This will cause the Storage daemon to maximize the use of drives.

Device Configuration Records

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

These four records, permitted in resources, are described in detail below. Note, however, that the **Changer Device** and the **Changer Command** directives are not needed in the resource if they are present in the **Autochanger** resource.

Autochanger = <yes|no> The **Autochanger** 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** . Note, some of the more advanced autochangers will locate the changer device on **/dev/sg1**. Such devices typically have several drives and a large number of tapes.

On FreeBSD systems, the changer device will typically be on **/dev/pass0** through **/dev/passn**.

On Solaris, the changer device will typically be some file under **/dev/rdsk**.

Please ensure that your Storage daemon has permission to access this device.

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
%d = changer drive index base 0
%f = Client's name
%i = JobId
%j = Job name
%l = archive control channel name
%o = command (loaded, load, or unload)
%s = Slot base 0
%S = Slot base 1
%v = Volume name
%V = Volume name from dcr->VolCatInfo first
```

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

```
Changer Command = "/opt/bacula/scripts/mtx-changer
```

Where you will need to adapt the **/opt/bacula** to be the actual path on your system where the **mtx-changer** script resides. 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 **5 minutes**. 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.

Drive Index = <number> This record allows you to tell **Bacula** to use the second or subsequent drive in an autochanger with multiple drives. Since the drives are numbered from zero, the second drive is defined by

```
Drive Index = 1
```

To use the second drive, you need a second Device resource definition in the **Bacula** configuration file. See the *Multiple Drive* section above in this chapter for more information.

In addition, for proper functioning of the Autochanger, you must define an Autochanger resource.

Note: If your tape hardware and operating system are relatively new (<10 years), they will cooperate smoothly.

See also:

Go back to:

- *Storage Resource*
- *Director Resource*
- *Device Resource*
- *Cloud Resource*

Go to:

- *Messages Resource*
- *Statistics Resource*

Go back to *Storage Daemon Resource Types* page.

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

Messages Resource

For a description of the Messages Resource, see the *Messages Resource* chapter.

See also:

Go back to:

- *Storage Resource*
- *Director Resource*
- *Device Resource*
- *Cloud Resource*
- *Autochanger Resource*

Go to:

- *Statistics Resource*

Go back to *Storage Daemon Resource Types* page.

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

Statistics Resource

The Resource defines the statistic collector function that can send information to a Graphite instance, to a CSV file or to bconsole with the **statistics** command (*See* for more information).

Statistics Start of the resource. Statistics directives are optional.

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

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

Interval = <time-interval> The Interval The directive instructs the Statistics collector thread how long it should sleep between every collection iteration. This directive is optional and the default value is **300** seconds.

Type = <CSV|Graphite> The Type The directive specifies the Statistics backend, which may be one of the following: **CSV** or **Graphite**. This directive is required.

CSV is a simple file level backend which saves all required metrics with the following format to the file: **"<time>, <metric>, <value> \n"**

Where **<time>** is a standard Unix time (a number of seconds from 01/01/1970) with local timezone as returned by a system call **time**(), **<metric>** is a Bacula metric string and is a metric value which could be in numeric format (int/float) or a string **True** or **False** for boolean variable. The CSV backend requires the parameter.

Graphite is a network backend which will send all required metrics to a Graphite server. The Graphite backend requires the **Host=** and **Port=** directives to be set.

If the Graphite server is not available, the metrics are automatically spooled in the working directory. When the server can be reached again, spooled metrics are despooled automatically and the spooling function is suspended.

Metrics = <metricspec> The **Metrics** directive allow metric filtering and is a filter which enables to use ***** and **?** characters to match the required metric name in the same way as found in shell wildcard resolution. You can exclude filtered metric with **!** prefix. You can define any number of filters for a single Statistics. Metrics filter is executed in order as found in configuration. This directive is optional and if not used all available metrics will be saved by this collector backend.

Example:

```
# Include all metric starting with "bacula.jobs"
Metrics = "bacula.jobs.*"

# Exclude any metric starting with "bacula.jobs"
Metrics = "!bacula.jobs.*"
```

Prefix = <string> The **Prefix** allows to alter the metrics name saved by collector to distinguish between different installations or daemons. This directive is optional. The prefix string will be added to metric name as: "<prefix>.<metric_name>"

File = <filename> The **File** is used by the CSV collector backend and point to the full path and filename of the file where metrics will be saved. With the CSV type, the **File** directive is required. The collector thread must have the permissions to write to the selected file or create a new file if the file doesn't exist. If collector is unable to write to the file or create a new one then the collection terminates and an error message will be generated. The file is only open during the dump and is closed otherwise. Statistics file rotation could be executed by a **mv** shell command.

Host = <hostname> The **Host** directive is used for Graphite backend and specify the hostname or the IP address of the Graphite server. When the directive Type is set to Graphite, the Host directive is required.

Host = <number> The **Port** directive is used for Graphite backend and specify the TCP port number of the Graphite server. When the directive Type is set to Graphite, the directive **Port** is required.

See also:

Go back to:

- *Storage Resource*
- *Director Resource*
- *Device Resource*
- *Cloud Resource*
- *Autochanger Resource*
- *Messages Resource*

Go back to *Storage Daemon Resource Types* page.

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

See an example of a Storage Daemon configuration file:

See also:

Go back to:

- *Director*

Go to:

- *File Daemon*
- *Console*

Go back to the *Technical Reference for Storage Daemon*.

Go back to the main *Technical Reference page*.

2.2 Example Storage Daemon Configuration File

An example Storage Daemon configuration file might be the following:

```
#
# Default Bacula Storage Daemon Configuration File
#
# For Bacula release 16.0.5 (05 April 2023) -- redhat (Blue
#
# You may need to change the name of your storage
# target in the Archive Device directive in the Device
# resources. If you change the Name and/or the Media
# Type in the Device resources, please ensure that
# bacula-dir.conf has the corresponding changes.
#
#
# Copyright (C) 2000-2020 Kern Sibbald
# License: BSD 2-Clause; see file LICENSE-FOSS
# ----------------------------------------------------

# This is a Storage Daemon definition. Here you can
# set the port that this Storage Daemon will listen on
```

(continues on next page)

(continued from previous page)

```
# and this needs to match the port defined in the
# Storage and Autochanger resources in the Director
# configuration file.
# -------------------------------------------------------
Storage {
  Name = bacula-sd
  SDPort = 9103
  WorkingDirectory = "/opt/bacula/working"
  PIDDirectory = "/opt/bacula/working"
  PluginDirectory = "/opt/bacula/plugins"
  MaximumConcurrentJobs = 20
}

# This is a Director resource. It defines a Director
# allowed to communicate with this Storage Daemon. The
# Password value specified in this Director resource
# must be the same value specified in the Password
# directive defined in the corresponding Storage or
# Autochanger resource in the Director configuration.
# -------------------------------------------------------
Director {
  Name = bacula-dir
  Password = "ZPUBve6cCwPVDpHqpqyAHPaPal30S/6LgSkpOzLrvLKU"
}

# This is an Autochanger resource. For Disk
# Autochangers, both Changer Command and Changer Device
# values must be blank and /dev/null respectively. For
# a Tape Autochanger or Tape Library definition, the
# mtx-changer script must be used.
# -------------------------------------------------------
Autochanger {
  Name = DiskAutochanger
  Device = DiskAutochanger_Dev0, DiskAutochanger_Dev1, DiskAutochanger_Dev2
  ChangerCommand = ""
  ChangerDevice = /dev/null
}

# The following three resources are Device resources.
# Each may be a single device or a member of an
# Autochanger resource. These devices are all members
# of the "DiskAutochanger" Autochanger resource defined
# above. The volumes managed by this device will be
# located in the /opt/bacula/archive directory. Please note
# that for disk and tape autochangers the DriveIndex
# directives must be used.
# -------------------------------------------------------
Device {
  Name = DiskAutochanger_Dev0
  DriveIndex = 0
  DeviceType = File
  MediaType = DiskVolume
```

(continues on next page)

(continued from previous page)

```
  ArchiveDevice = "/opt/bacula/archive"
  LabelMedia = Yes
  RandomAccess = Yes
  AutomaticMount = Yes
  RemovableMedia = No
  AlwaysOpen = No
  MaximumConcurrentJobs = 5
}

Device {
  Name = DiskAutochanger_Dev1
  DriveIndex = 1
  DeviceType = File
  MediaType = DiskVolume
  ArchiveDevice = "/opt/bacula/archive"
  LabelMedia = Yes
  RandomAccess = Yes
  AutomaticMount = Yes
  RemovableMedia = No
  AlwaysOpen = No
  MaximumConcurrentJobs = 5
}

Device {
  Name = DiskAutochanger_Dev2
  DriveIndex = 2
  DeviceType = File
  MediaType = DiskVolume
  ArchiveDevice = "/opt/bacula/archive"
  LabelMedia = Yes
  RandomAccess = Yes
  AutomaticMount = Yes
  RemovableMedia = No
  AlwaysOpen = No
  MaximumConcurrentJobs = 5
  Autoselect = No
}

# This is a single, stand alone Device resource. It is
# used by the Catalog Backup job and will help keep
# Catalog backups separated from all other backup data
# (different MediaType).
# -------------------------------------------------------
Device {
  Name = DiskCatalogDevice
  DeviceType = File
  MediaType = CatalogVolume
  ArchiveDevice = "/opt/bacula/archive"
  LabelMedia = Yes
  RandomAccess = Yes
  AutomaticMount = Yes
  RemovableMedia = No
```

(continues on next page)

(continued from previous page)

```
  AlwaysOpen = No
  MaximumConcurrentJobs = 1
}

# This is a Messages resource. All the messages
# generated by this Storage Daemon will be sent to the
# Director specified in this Messages resource.
# ----------------------------------------------------
Messages {
  Name = Default
  Director = bacula-dir = all
  Append = "/opt/bacula/log/bacula-sd.log" = All, !Skipped
}
```

See also:

Go back to *Storage Daemon Resource Types*.

Go back to the *Technical Reference for Storage Daemon*.

See also:

Go back to:

- *Director*

  Go to:

  - *File Daemon*
  - *Console*

Go back to the main *Technical Reference page*.

3 File Daemon

The File Daemon (or Client) 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**, see the Configuration chapter of this manual.

Read more:

3.1 File Daemon (Client) Resource Types

The following File Daemon (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.

Client Resource

The Client Resource (or FileDaemon) resource defines the name of the Client (as used by the Director) as well as the port on which the Client listens for Director connections.

Client (or FileDaemon) Start of the Client records. There must be one and only one Client resource in the configuration file, since it defines the properties of the current client program.

Name = <name> The client name that must be used by the Director when connecting. Generally, it is a good idea to use a name related to the machine so that error messages can be easily identified if you have multiple Clients. This directive is required.

Working Directory = <Directory> This directive is mandatory and specifies a directory in which the File daemon may put its status files. This directory should be used only by **Bacula**, but may be shared by other daemons provided the daemon names on the definition are unique for each daemon. This directive is required.

Pid Directory = <Directory> This directive is mandatory and specifies a directory in which the File daemon may put its process Id file files. The process Id file is used to shutdown **Bacula** and to prevent multiple copies of from running simultaneously. This record is required. Standard shell expansion of the 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.

ClientRehydration = <Directory> This directive is optional and allows to try to do rehydration using existing local data on the Client at restore time. In some cases, the use of this directive permits to transfer less data over the network during a restore. The default value is no.

Heartbeat Interval = <time-interval> This record defines an interval of time in seconds. For each heartbeat that the File daemon receives from the Storage daemon, it will forward it to the Director. In addition, if no heartbeat has been received from the Storage daemon and thus forwarded the File daemon will send a heartbeat signal to the Director and to the Storage daemon to keep the channels active. The default interval is 300s. This feature is particularly useful if you have a router that does not follow Internet standards and times out a valid connection after a short duration despite the fact that keepalive is set. This usually results in a broken pipe error message.

If you continue getting broken pipe error messages despite using the Heartbeat Interval, and you are using Windows, you should consider upgrading your ethernet driver.

Lack of communications, or communications that get interrupted can also be caused by Linux firewalls where you have a rule that throttles connections or traffic.

Maximum Concurrent Jobs = <number> where <number> the maximum number of Jobs that should run concurrently. The default is set to **20**, 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. If set to a large value, please be careful to have this value higher than the **Maximum Concurrent Jobs** configured in the resource in the Director configuration file. Otherwise, backup jobs can fail due to the Director connection to FD be refused because MCJ was exceeded on FD side.

Maximum Job Error Count = <number> where <number> is the error threshold for the Job, after reaching it Job will be failed. The default value is 1000. If this value is set to 0, job will continue to run no matter how many errors it encounters.

FDAddresses = <IP-address-specification> Specify the ports and addresses on which the File daemon listens for Director connections. Probably the simplest way to explain is to show an example:

```
FDAddresses = {
    ip = { addr = 1.2.3.4; port = 1205; }
    ipv4 = {
```

(continues on next page)

(continued from previous page)

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

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

FDPort = <port-number> This specifies the port number on which the Client listens for Director connections. It must agree with the FDPort specified in the Client resource of the Director's configuration file. The default is **9102**.

FDAddress = <IP-Address> This record is optional, and if it is specified, it will cause the File daemon server (for Director connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this record is not specified, the File daemon will bind to any available address (the default).

FDSourceAddress = <IP-Address> This record is optional, and if it is specified, it will cause the File daemon server (for Storage connections) to bind to the specified **IP-Address**, which is either a domain name or an IP address specified as a dotted quadruple. If this record is not specified, the kernel will choose the best address according to the routing table (the default).

SDConnectTimeout = <time-interval> This record defines an interval of time that the File daemon will try to connect to the Storage daemon. The default is **30 minutes**. If no connection is made in the specified time interval, the File Daemon cancels the Job.

Maximum Network Buffer Size = <bytes> where specifies the initial network buffer size to use with the File daemon. This size will be adjusted down if it is too large until it is accepted by the OS. Please use care in setting this value since if it is too large, it will be trimmed by 512 bytes until the OS is happy, which may require a large number of system calls. The default value is **65,536** bytes. The maximum value is **1,000,000** bytes.

Note: On certain Windows machines, there are reports that the transfer rates are very slow and this seems to be related to the default **65,536** size. On systems where the transfer rates seem abnormally slow compared to other systems, you might try setting the Maximum Network Buffer Size to 32,768 in both the File daemon and in the Storage Daemon.

Maximum Bandwidth Per Job = <speed> The speed parameter specifies the maximum allowed bandwidth in bytes per second that a job may use. You may specify the following speed parameter modifiers: **kb/s** (1,000 bytes per second), **k/s** (1,024 bytes per second), **mb/s** (1,000,000 bytes per second), or **m/s** (1,048,576 bytes per second).

The use of TLS, TLS PSK, CommLine compression and Deduplication can interfere with the value set with this Directive.

CommCompression = <yes|no> If the two **Bacula** components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes.

In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is **enabled**. In the case that the compression is not effective, **Bacula** turns it off on a record by record basis.

If you are backing up data that is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, **Bacula** reports **None** in the Job report.

DisableCommand = <cmd> The **Disable Command** adds security to your File Daemon by disabling certain commands globally. The commands that can be disabled are:

```
backup
cancel
setdebug=
setbandwidth=
estimate
fileset
JobId=
level=
restore
endrestore
session
status
.status
storage
verify
RunBeforeNow
RunBeforeJob
RunAfterJob
Run
accurate
```

One or more of these command keywords can be placed in quotes and separated by spaces on the directive line. Note: the commands must be written exactly as they appear above.

SD Packet Check = <num-packets> The **SDPacketCheck** takes a positive integer. If the integer is zero, it turns off this feature. If the integer is greater than zero, it is the number of packets that the FileDaemon will send to the Storage Daemon before sending a POLL request and waiting for the Storage Daemon answer. The default value is 0. If the time between two POLL requests is too short (less than few seconds) and the number of bytes transferred is less than few hundred of MB, the value of the **SDPacketCheck** is increased dynamically.

FIPS Require = <yes|no> Require FIPS cryptographic module to start the daemon.

Note: TLS Directives in the FileDaemon (or Client) resource of bacula-fd.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh**. The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

Plugin Directory = <Directory>

Each daemon (DIR, FD, SD) has a **Plugin Directory** directive that may be added to the daemon definition resource. The directory takes a quoted string argument, which is the name of the directory in which the daemon can find the Bacula plugins. If this directive is not specified, Bacula will not load any plugins. Since each plugin has a distinctive name, all the daemons can share the same plugin directory.

Plugin Options = <Plugin-Command-Line>

To configure non-job specific plugins (antivirus, security, ...), it is possible to use the **PluginOptions** directive. Multiple directives are allowed.

PKI Cipher This directive is optional and if specified will configure the data encryption to use a specific cipher. The default cipher is **AES 128 CBC**.

The following ciphers are available: aes128, aes192, aes256 and blowfish.

PKI Digest This directive is optional and if specified will configure the data encryption to use a specific digest algorithm. The default cipher is **SHA!** or **sha256** depending on the version of OpenSSL.

The following digest are available: md5, sha1, sha256.

PKI Encryption See the Data Encryption chapter.

PKI Signatures See the Data Encryption chapter.

PKI Keypair See the Data Encryption chapter.

PKI Master Key See the Data Encryption chapter.

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

```
Client {
  Name = rufus-fd
  WorkingDirectory = $HOME/bacula/bin/working
  PIDDirectory = $HOME/bacula/bin/working
}
```

See also:

Go to:

- *Director Resource*
- *Schedule Resource*
- *Messages Resource*
- *Statistics Resource*

Go back to the *File Daemon (Client) Resource Types* page.

Go back to the *Technical Reference for File Daemon*.

Go back to the main *Technical Reference page*.

Director Resource

The Director resource defines the name and password of the Directors that are permitted to contact this Client.

Director Start of the Director records. There may be any number of Director resources in the Client configuration file. Each one specifies a Director that is allowed to connect to this Client.

Name = <name> The name of the Director that may contact this Client. This name must be the same as the name specified on the Director resource in the Director's configuration file. Note, the case (upper/lower) of the characters in the name are significant (i.e. S is not the same as s). This directive is required.

Password = <password> Specifies the password that must be supplied for a Director to be authorized. This password must be the same as the password specified in the Client resource in the Director's configuration file. This directive is required.

DirPort = <number> Specify the port to use to connect to the Director. This port must be identical to the **DIRport** specified in the **Director** resource of the Director's configuration file. The default **9101** is so this directive is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director. This directive is required when **ConnectToDirector** is enabled.

ConnectToDirector = <yes|no> When the **ConnectToDirector** directive is set to `true`, the Client will contact the Director according to the rules. The connection initiated by the Client will be then used by the Director to start jobs or issue bconsole commands. If the **Schedule** directive is not set, the connection will be initiated when the file daemon starts. The connection will be reinitialized every **ReconnectionTime**. This directive can be useful if your File Daemon is behind a firewall that permits outgoing connections but not incoming connections.

ReconnectionTime = <time> When the Director resource of the FileDaemon is configured to connect the Director with the **ConnectToDirector** directive, the connection initiated by the FileDeamon to the Director will be reinitialized at a regular interval specified by the **ReconnectionTime** directive. The default value is **40 mins**.

Schedule = <sched-resource> The directive defines what schedule is to be used for Client to connect the Director if the directive is set to `true`. This directive is optional, and if left out, the Client will initiate a connection automatically at the start of the daemon. Although you may specify only a single Schedule resource for any resource, the resource may contain multiple **Connect** directives, which allow you to initiate the Client connection at many different times, and each **Connect** directive permits to set the **Max Connect Time** directive.

Maximum Bandwidth Per Job = <speed> The speed parameter specifies the maximum allowed bandwidth in bytes per second that a job may use when started from this Director. You may specify the following speed parameter modifiers: kb/s (1,000 bytes per second), k/s (1,024 bytes per second), mb/s (1,000,000 bytes per second), or m/s (1,048,576 bytes per second).

DisableCommand = <cmd> The **Disable Command** adds security to your File daemon by disabling certain commands for the current Director. More information about the syntax can be found in the DisableCommand section.

Monitor = <yes|no> If Monitor is set to **no** (default), this director will have full access to this Client. If Monitor is set to yes, this director will only be able to fetch the current status of this Client. Please note that if this director is being used by a Monitor, we highly recommend to set this directive to **yes** to avoid serious security problems.

Backup Job User = <uid:gid> Lets the daemon control in which user's context the Backup job is being run. Can be expressed as uid:gid or username:groupname.

Restore Job User = <uid:gid> Lets the daemon control in which user's context the Restore job is being run. It can be expressed as uid:gid or username:groupname. Overrides the Restore user set with 'jobuser' and 'jobgroup' arguments for the bconsole's 'restore' command.

Allowed Backup Directories = <Directories list> Defines per- Director list of client's directories that are allowed to be backed up for a specific Director. This directive is fully independent of the include/exclude part of the Fileset defined in the Director's config file. Nothing is backed up if none of the files defined in the Fileset is inside FD's allowed directory. This directive is not required. If it is not defined, every directory defined in the Fileset is allowed.

Excluded Backup Directories = <Directories list> Defines list of client's directories per Director that are excluded from the backup for specific Director. This directive is fully independent of the include/exclude part of the Fileset defined in the Director's config file. Nothing is backed up if all files defined in the Fileset are inside FD's excluded directory.

Allowed Restore Directories = <Directories list> Defines per-Director list of client's directories that are allowed to be used as a restore destination on a per-Director basis. Directive can be specified as a list of directories.

Allowed Script Directories = <Directories list> Defines per- Director list of client's directories which from the Director can execute client's scripts and programs (e.g. using the Runscript feature or with the Fileset's 'File=' directive). Directive can be specified as a list of directories. When this directive is set, Bacula is also checking programs to be run against set of not-allowed characters.Full list of not-allowed characters:

```
$ ! ; \ & < > ` ( )
```

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 multiple Director resource definitions:

```
#
# List Directors who are permitted to contact the File daemon
#
Director {
    Name = HeadMan
    Password = very_good # password HeadMan must supply
}
Director {
    Name = Worker
    Password = not_as_good
    Monitor = Yes
}
```

Note: TLS Directives in the Director resource of bacula-fd.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh** . The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

See also:

Go back to:

- *Client Resource*

Go to:

- *Schedule Resource*
- *Messages Resource*
- *Statistics Resource*

Go back to the *File Daemon (Client) Resource Types* page.

Go back to the *Technical Reference for File Daemon*.

Go back to the main *Technical Reference page*.

Schedule Resource

The Schedule resource provides a means of automatically scheduling the connection of the Client to the Director. Each Director resource can have a different Schedule resource.

Schedule Start of the Schedule directives. No resource is required. If no is used, the Client will connect automatically to the Director at the startup.

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

Enabled = <yes|no> This directive allows you to enable or disable the resource.

Connect = <Connect-overrides> <Date-time-specification> The Connect directive defines when a Client should connect to a Director. You may specify multiple **Connect** directives within a resource. If you do, they will all be applied (i.e. multiple schedules). If you have two **Connect** directives that start at the same time, two connections will start at the same time (well, within one second of each other). It is not recommended to have multiple connections at the same time.

Connect-options are specified as: **keyword=value** where the keyword is **MaxConnectTime** and the **value** is as defined on the respective directive formats for the Job resource. You may specify multiple **Connect-options** on one directive by separating them with one or more spaces or by separating them with a trailing comma. For example:

MaxConnectTime=<time-spec> specifies how much time the connection will be attempt and active.

The <Date-time-specification> is similar to what exists for Job scheduling. See **Date-time-specification** for more information.

An example schedule resource that is named **WeeklyCycle** and connects a director each Sunday at 2:05am and an on Monday through Saturday at 2:05am is:

```
Schedule {
  Name = "WeeklyCycle"
  Connect = MaxConnectTime=2h sun at 2:05
  Connect = MaxConnectTime=2h mon-sat at 2:05
}
```

In this example, the Job for this client should be schedule to start between 2:05 and 4:05 during the week. The **Bacula** administrator will be able to use commands such as **status client** or **estimate** betwen 2:05 and 4:05.

See also:

Go back to:

- *Client Resource*
- *Director Resource*

Go to:

- *Messages Resource*
- *Statistics Resource*

Go back to the *File Daemon (Client) Resource Types* page.

Go back to the *Technical Reference for File Daemon*.

Go back to the main *Technical Reference page*.

Messages Resource

See the *Messages Resource* chapter for the details of the Messages Resource.

There must be at least one Messages resource in the Client configuration file.

See also:

Go back to:

- *Client Resource*
- *Director Resource*
- *Schedule Resource*

Go to:

- *Statistics Resource*

Go back to the *File Daemon (Client) Resource Types* page.

Go back to the *Technical Reference for File Daemon*.

Go back to the main *Technical Reference page*.

Statistics Resource

The Resource defines the statistic collector function that can send information to a Graphite instance, to a `CSV` file or **bconsole** to with the **statistics** command (See *Daemon Real-Time Statistics Monitoring* for more information).

Statistics Start of the resource. Statistics directives are optional.

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

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

Interval = <time-interval> The **Intervall** directive instructs the Statistics collector thread how long it should sleep between every collection iteration. This directive is optional and the default value is **300** seconds.

Type = <CSV|Graphite> The **Type** directive specifies the Statistics backend, which may be one of the following: CSV or **Graphite**. This directive is required.

`CSV` is a simple file level backend which saves all required metrics with the following format to the file: "<time>, <metric>, <value> \n"

Where **<time>** is a standard Unix time (a number of seconds from 1/01/1970) with local timezone as returned by a system call **time()**, **<metric>** is a **Bacula** metric string and is a metric value which could be in numeric format (`int/float`) or a string **"True"** or **"False"** for a boolean variable. The `CSV` backend requires the **file =** parameter.

Graphite is a network backend which will send all required metrics to a Graphite server. The Graphite backend requires the **Host=** and **Port=** directives to be set.

If the Graphite server is not available, the metrics are automatically spooled in the working directory. When the server can be reached again, spooled metrics are despooled automatically and the spooling function is suspended.

Metrics = <metricspec> The **Metrics** directive allow metric filtering and is a filter which enables to use ***** and **?** characters to match the required metric name in the same way as found in shell wildcard resolution. You can exclude filtered metric with **!** prefix. You can define any number of filters for a single Statistics. Metrics filter is executed in order as found in configuration. This directive is optional and if not used all available metrics will be saved by this collector backend.

Example:


```
# Include all metric starting with "bacula.jobs"
Metrics = "bacula.jobs.*"

# Exclude any metric starting with "bacula.jobs"
Metrics = "!bacula.jobs.*"
```

Prefix = <string> The **Prefix** allows to alter the metrics name saved by collector to distinguish between different installations or daemons. The prefix string will be added to metric name as: "<prefix> . <metric_name>" This directive is optional.

File = <filename> The **File** is used by the CSV collector backend and point to the full path and filename of the file where metrics will be saved. With the CSV type, the **File** directive is required. The collector thread must have the permissions to write to the selected file or create a new file if the file doesn't exist. If collector is unable to write to the file or create a new one then the collection terminates and an error message will be generated. The file is only open during the dump and is closed otherwise. Statistics file rotation could be executed by a mv shell command.

Host = <hostname> The **Host** directive is used for Graphite backend and specify the hostname or the IP address of the Graphite server. When the directive Type is set to Graphite, the **Host** directive is required.

Port = <number> The **Port** directive is used for Graphite backend and specify the TCP port number of the Graphite server. When the directive Type is set to Graphite, the **Port** directive is required.

See also:

Go back to:

- *Client Resource*
- *Director Resource*
- *Schedule Resource*
- *Messages Resource*

Go back to the *File Daemon (Client) Resource Types* page.

Go back to the *Technical Reference for File Daemon*.

Go back to the main *Technical Reference page*.

See an example of a File Daemon configuration file:

See also:

Go back to:

- *Director*
- *Storage Daemon*

Go to:

- *Console*

Go back to the *Technical Reference for File Daemon*.

Go back to the main *Technical Reference page*.

3.2 Example File Daemon Configuration File

An example File Daemon configuration file might be the following:

```
# Default Bacula File Daemon Configuration file
#
# For Bacula release 1.35.2 (16 August 2004) -- gentoo 1.4.16
#
# There is not much to change here except perhaps to
#
# set the Director's name and File daemon's name
#
# to something more appropriate for your site.
#

# List Directors who are permitted to contact this File daemon
Director {
    Name = rufus-dir
    Password = "/LqPRkX++saVyQE7w7mmiFg/qxYc1kufww6FEyY/47jU"
}

# Restricted Director, used by tray-monitor to get the
#
# status of the file daemon
Director {
    Name = rufus-mon
    Password = "FYpq4yyI1y562EMS35bA0J0QC0M2L3t5cZObxT3XQxgxppTn"
    Monitor = yes
}

# "Global" File daemon configuration specifications
FileDaemon {
    # this is me
    Name = rufus-fd
    WorkingDirectory = $HOME/bacula/bin/working
    Pid Directory = $HOME/bacula/bin/working
}

# Send all messages except skipped files back to Director
Messages {
    Name = Standard
    director = rufus-dir = all, !skipped
}
```

See also:

Go back to *File Daemon Resource Types*.

Go back to the Technical Reference for File Daemon.

See also:

Go back to:

- *Director*
- *Storage Daemon*

Go to:

- *Console*

Go back to the main *Technical Reference page*.

4 Console

The Console configuration file is the simplest of all the configuration files, and in general, you should not need to change it except for the password. It simply contains the information necessary to contact the Director or Directors.

For a general discussion of the syntax of configuration files and their resources including the data types recognized by **Bacula**, see the Configuration chapter.

Read more:

4.1 Console Resource Types

The following Console Resource definition must be defined:

Director Resource

The Director resource defines the attributes of the Director running on the network. You may have multiple Director resource specifications in a single Console configuration file. If you have more than one, you will be prompted to choose one when you start the **Console** program.

Director Start of the Director directives.

Name = <name> The director name used to select among different Directors, otherwise, this name is not used.

DIRPort = <port-number> Specify the port to use to connect to the Director. This value will most likely already be set to the value you specified on the ``–``**with-baseport** option of the **./configure** command. This port must be identical to the **DIRport** specified in the **Director** resource of the Director's configuration file. The default is 9101 so this directive is not normally specified.

Address = <address> Where the address is a host name, a fully qualified domain name, or a network address used to connect to the Director.

Password = <password> Where the password is the password needed for the Director to accept the Console connection. This password must be identical to the **Password** specified in the **Director** resource of the file. This directive is required.

HistoryFile = <filename> Where the filename will be used to store the console command history. By default, the history file is set to `$HOME/.bconsole_history`

HistoryFileSize = <number-of-lines> Specify the history file size in lines. The default value is 100.

FIPS Require = <yes|no> Require **FIPS!** cryptographic module to start the daemon.

Note: TLS Directives in the Director resource of bconsole.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh** . The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

An actual example might be:


```
Director {
    Name = bacula-dir
    address = baculadir.example.com
    password = xyz1erploit
}
```

See also:

Go to:

- *ConsoleFont Resource*
- *Console Resource*

Go back to *Console Resource Types* page.

Go back to the *Technical Reference for Console Daemon*.

Go back to the main *Technical Reference page*.

ConsoleFont Resource

The ConsoleFont resource is available only in the GNOME version of the console. It permits you to define the font that you want used to display in the main listing window.

ConsoleFont Start of the ConsoleFont directives.

Name = <name> The name of the font.

Font = <Pango Font Name> The string value given here defines the desired font. It is specified in the Pango format. For example, the default specification is:

```
Font = "LucidaTypewriter 9"
```

An different example might be:

```
ConsoleFont {
    Name = Default
    Font = "Monospace 10"
    }
```

See also:

Go back to:

- *Director Resource*

Go to:

- *Console Resource*

Go back to *Console Resource Types* page.

Go back to the *Technical Reference for Console Daemon*.

Go back to the main *Technical Reference page*.

Console Resource

There are three different kinds of consoles, which the administrator or user can use to interact with the Director. These three kinds of consoles comprise three different security levels.

- The first console type is an **anonymous** or **default** console, which has full privileges. There is no console resource necessary for this type since the password is specified in the Director resource. Typically you would use this **anonymous** console only for administrators. The second type of console is a "named" or "restricted" console defined within a Console resource in both the Director's configuration file and in the Console's configuration file. Both the names and the passwords in these two entries must match much as is the case for Client programs.
- This second type of console begins with absolutely no privileges except those explicitly specified in the Director's Console resource. Note, the definition of what these restricted consoles can do is determined by the Director's conf file. Thus you may define within the Director's conf file multiple Consoles with different names and passwords, sort of like multiple users, each with different privileges. As a default, these consoles can do absolutely nothing – no commands what so ever. You give them privileges or rather access to commands and resources by specifying access control lists in the Director's Console resource. This gives the administrator fine grained control over what particular consoles (or users) can do.
- The third type of console is similar to the above mentioned restricted console in that it requires a Console resource definition in both the Director and the Console. In addition, if the console name, provided on the **Name =** directive, is the same as a Client name, the user of that console is permitted to use the **SetIP** command to change the Address directive in the Director's client resource to the IP address of the Console. This permits portables or other machines using DHCP (non-fixed IP addresses) to "notify" the Director of their current IP address.

The Console resource is optional and need not be specified. However, if it is specified, you can use ACLs (Access Control Lists) in the Director's configuration file to restrict the particular console (or user) to see only information pertaining to his jobs or client machine.

You may specify as many Console resources in the console's conf file. If you do so, generally the first Console resource will be used. However, if you have multiple Director resources (i.e. you want to connect to different directors), you can bind one of your Console resources to a particular Director resource, and thus when you choose a particular Director, the appropriate Console configuration resource will be used. See the "Director" directive in the Console resource described below for more information.

Note: The Console resource is optional, but can be useful for restricted consoles as noted above.

Console Start of the Console resource.

Name = <name> The Console name used to allow a restricted console to change its IP address using the SetIP command. The SetIP command must also be defined in the Director's conf CommandACL list.

Password = <password> If this password is supplied, then the password specified in the Director resource of you Console conf will be ignored. See below for more details.

Director = <director-resource-name> If this directive is specified, this Console resource will be used by bconsole when that particular director is selected when first starting bconsole. I.e. it binds a particular console resource with its name and password to a particular director.

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

CommCompression = <yes|no> If the two Bacula components (DIR, FD, SD, bconsole) have the comm line compression enabled, the line compression will be enabled. The default value is yes. In many cases, the volume of data transmitted across the communications line can be reduced by a factor of three when this directive is **enabled**. In the case that the compression is not effective, Bacula turns it off on a record by record basis. If you are backing up data that

is already compressed the comm line compression will not be effective, and you are likely to end up with an average compression ratio that is very small. In this case, Bacula reports **None** in the Job report.

Note: TLS Directives in the Console resource of bconsole.conf

Bacula has built-in network encryption code to provide secure network transport similar to that offered by **stunnel** or **ssh** . The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code.

For more information how to enable TLS encryption, click here.

For example, if we define the following in the user's **bconsole.conf** file (or perhaps the **bwx-console.conf** file):

```
Director {
    Name = bacula-dir
    DIRport = 9101
    Address = baculadir.example.com
    Password = "XXXXXXXXXXX" # no, really. this is not obfuscation.
}

Console {
    Name = restricted-user
    Password = "UntrustedUser"
}
```

Where the Password in the Director section is deliberately incorrect, and the Console resource is given a name, in this case **restricted-user**. Then in the Director's **bacula-dir.conf** file (not directly accessible by the user), we define:

```
Console {
    Name = restricted-user
    Password = "UntrustedUser"
    JobACL = "Restricted Client Save"
    ClientACL = restricted-client
    StorageACL = main-storage
    ScheduleACL = *all*
    PoolACL = *all*
    FileSetACL = "Restricted Client's FileSet"
    CatalogACL = BaculaCatalog
    CommandACL = run
}
```

the user logging into the Director from his Console will get logged in as **restricted-user**, and he will only be able to see or access a Job with the name **Restricted Client Save** a Client with the name **restricted-client**, a Storage device **main-storage**, any Schedule or Pool, a FileSet named **Restricted Client's FileSet**, a Catalog named **DefaultCatalog**, and the only command he can use in the Console is the **run** command. In other words, this user is rather limited in what he can see and do with Bacula.

The following is an example of a bconsole conf file that can access several Directors and has different Consoles depending on the director:

```
Director {
    Name = bacula1-dir
    DIRport = 9101
    Address = bacula1.example.com
```

(continues on next page)

(continued from previous page)

```
    Password = "XXXXXXXXXXX" # no, really. this is not obfuscation.
}
Director {
    Name = bacula2-dir
    DIRport = 9101
    Address = bacula2.example.com
    Password = "XXXXXXXXXXX" # no, really. this is not obfuscation.
}

Console {
    Name = restricted-user
    Password = "UntrustedUser"
    Director = bacula1-dir
}
Console {
    Name = restricted-user-2
    Password = "A different UntrustedUser"
    Director = bacula2-dir
}
```

The second Director referenced at “bacula2-dir” might look like the following:

```
Console {
    Name = restricted-user-2
    Password = "A different UntrustedUser"
    JobACL = "Restricted Client Save"
    ClientACL = restricted-client
    StorageACL = second-storage
    ScheduleACL = *all*
    PoolACL = *all*
    FileSetACL = "Restricted Client's FileSet"
    CatalogACL = Bacula2Catalog
    CommandACL = run, restore
    WhereACL = "/"
}
```

To use the same Console name on both Directors, you must create two **bconsole.conf** to store the two Director/Console groups.

Console Commands

For more details on running the console and its commands, see the Bacula Console chapter.

See also:

Go back to:

- *Director Resource*
- *ConsoleFont Resource*

Go back to *Console Resource Types* page.

Go back to the *Technical Reference for Console Daemon*.

Go back to the main *Technical Reference page*.

See an example of a Console configuration file:

See also:

Go back to:

- *Director*
- *Storage Daemon*
- *File Daemon*

Go back to the *Technical Reference for Console Daemon*.

Go back to the main *Technical Reference page*.

4.2 Example Console Configuration File

An example Console configuration file might be the following:

```
#
# Bacula Console Configuration File
#
Director {
    Name = bacula-dir
    address = "baculadir.example.com"
    Password = Console_password
}
```

See also:

Go back to *Console Resource Types*.

Go back to the *Technical Reference for Console*.

See also:

Go back to:

- *Director*
- *Storage Daemon*
- *File Daemon*

Go back to the main *Technical Reference page*.

5 Variable Expansion

Variable expansion is somewhat similar to Unix shell variable expansion. It is used only in format labels.

5.1 General Functionality

This is basically a string expansion capability that permits referencing variables, indexing arrays, conditional replacement of variables, case conversion, substring selection, regular expression matching and replacement, character class replacement, padding strings, repeated expansion in a user controlled loop, support of arithmetic expressions in the loop start, step and end conditions, and recursive expansion.

When using variable expansion characters in a Volume Label Format record, the format should always be enclosed in double quotes (**"**).

For example, **?{HOME}** will be replaced by your home directory as defined in the environment. If you have defined the variable **xxx** to be **Test**, then the reference **?{xxx:p/7/Y/r}** will right pad the contents of **xxx** to a length of seven characters filling with the character **Y** giving **YYYTest**.

5.2 Bacula Variables

Within **Bacula**, there are three main classes of variables with some minor variations within the classes. The classes are:

Counters

Counters are defined by the **Counter** resources in the Director's conf file. The counter can either be a temporary counter that lasts for the duration of **Bacula**'s execution, or it can be a variable that is stored in the catalog, and thus retains its value from one **Bacula** execution to another. Counter variables may be incremented by postfixing a plus sign (**+** after the variable name).

Internal Variables

Internal variables are read-only, and may be related to the current job (i.e. Job name), or maybe special variables such as the date and time.

The following variables are available:

- Year – the full year
- Month – the current month 1-12
- Day – the day of the month 1-31
- Hour – the hour 0-24
- Minute – the current minute 0-59
- Second – the current second 0-59
- WeekDay – the current day of the week 0-6 with 0 being Sunday
- Job – the job name
- general – the Director's name
- Level – the Job Level
- Type – the Job type
- JobId – the JobId
- JobName – the unique job name composed of Job and date
- JobTimestamp – the Job timestamp

- Storage – the Storage daemon's name
- Client – the Client's name
- NumVols – the current number of Volumes in the Pool
- Pool – the Pool name
- Catalog – the Catalog name
- MediaType – the Media Type
- PriorJobName – the prior Job name (for copy and migration jobs)
- PriorJobId – the prior JobId (for copy and migration jobs)

Environment Variables

Environment variables are read-only, and must be defined in the environment prior to executing **Bacula**. Environment variables may be either scalar or an array, where the elements of the array are referenced by subscripting the variable name (e.g. **?{Months[3]}**). Environment variable arrays are defined by separating the elements with a vertical bar (|), thus **set Months="Jan|Feb|Mar|Apr|..."** defines an environment variable named **Month** that will be treated as an array, and the reference **?{Months[3]}** will yield **Mar**. The elements of the array can have differing lengths.

5.3 Full Syntax

Since the syntax is quite extensive, below, you will find the pseudo BNF. The special characters have the following meaning:

```
::= definition
( ) grouping if the parens are not quoted
| separates alternatives
'/' literal / (or any other character)
CAPS a character or character sequence
* preceding item can be repeated zero or more times
? preceding item can appear zero or one time
+ preceding item must appear one or more times
```

And the pseudo BNF describing the syntax is:

```
input ::= ( TEXT
                | variable
                | INDEX_OPEN input INDEX_CLOSE (loop_limits)?
                )*
variable ::= DELIM_INIT (name|expression)
name ::= (NAME_CHARS)+
expression ::= DELIM_OPEN
                (name|variable)+
                (INDEX_OPEN num_exp INDEX_CLOSE)?
                (':' command)*
                DELIM_CLOSE
command ::= '-' (TEXT_EXP|variable)+
                | '+' (TEXT_EXP|variable)+
                | 'o' NUMBER ('-'|',') (NUMBER)?
```

(continues on next page)

(continued from previous page)

```
                | '#'
                | '*' (TEXT_EXP|variable)+
                | 's' '/' (TEXT_PATTERN)+
                        '/' (variable|TEXT_SUBST)*
                        '/' ('m'|'g'|'i'|'t')*
                | 'y' '/' (variable|TEXT_SUBST)+
                        '/' (variable|TEXT_SUBST)*
                        '/'
                | 'p' '/' NUMBER
                        '/' (variable|TEXT_SUBST)*
                        '/' ('r'|'l'|'c')
                | '%' (name|variable)+
                        ('(' (TEXT_ARGS)? ')')?
                | 'l'
                | 'u'
num_exp ::= operand
                | operand ('+'|'-'|'*'|'/'|'%') num_exp
operand ::= ('+'|'-')? NUMBER
                | INDEX_MARK
                | '(' num_exp ')'
                | variable
loop_limits ::= DELIM_OPEN
                (num_exp)? ',' (num_exp)? (',' (num_exp)?)?
                DELIM_CLOSE
NUMBER       ::= ('0'|...|'9')+
TEXT_PATTERN ::= (^('/'))+
TEXT_SUBST   ::= (^(DELIM_INIT|'/'))+
TEXT_ARGS    ::= (^(DELIM_INIT|')'))+
TEXT_EXP         ::= (^(DELIM_INIT|DELIM_CLOSE|':'|'+'))+
TEXT             ::= (^(DELIM_INIT|INDEX_OPEN|INDEX_CLOSE))+
DELIM_INIT   ::= '$'
DELIM_OPEN   ::= '{'
DELIM_CLOSE  ::= '}'
INDEX_OPEN   ::= '['
INDEX_CLOSE  ::= ']'
INDEX_MARK   ::= '#'
NAME_CHARS   ::= 'a'|...|'z'|'A'|...|'Z'|'0'|...|'9'
```

5.4 Semantics

The items listed in **command** above, which always follow a colon (**:**) have the following meanings:

```
- perform substitution if variable is empty
+ perform substitution if variable is not empty
o cut out substring of the variable value
# length of the variable value
* substitute empty string if the variable value is not empty,
otherwise substitute the trailing parameter
s regular expression search and replace. The trailing
options are: m = multiline, i = case insensitive,
g = global, t = plain text (no regexp)
```

(continues on next page)

(continued from previous page)

```
y transpose characters from class A to class B
p pad variable to l = left, r = right or c = center,
with second value.
% special function call (none implemented)
l lower case the variable value
u upper case the variable value
```

The **loop_limits** are start, step, and end values.

A counter variable name followed immediately by a plus (**+**) will cause the counter to be incremented by one.

5.5 Examples

To create an ISO date:

```
DLT-$Year-$Month:p/2/0/r-$Day:p/2/0/r
```

on 20 June 2022 would give **DLT-2022-06-20**

If you set the environment variable **mon** to

```
January|February|March|April|May|... File-$mon[$Month]/$Day/$Year
```

on the first of March would give **File-March/1/2022**

