Firebird Backup & Restore Utility

Norman Dunbar

1 may 2013 - Document version 1.11

Table of Contents

Introduction

Gbak is one of the database backup and restore utilities supplied with Firebird. In Firebird 1.5 it is the only supplied utility of this kind while Firebird 2.x also has the nbackup utility which is described in another document.

In this manual, we will discuss:

- Command-line options for gbak.
- gbak commands and their parameters.
- Running gbak in backup or restore modes.
- Some caveats, gotchas and foibles of gbak.

Command-line Options

Common Options

When running gbak in backup or restore mode, there are a number of options which apply to either mode. These are:

- **`-?`**

  This switch displays the command line options and switches. It replaces the old method in which you had to supply an invalid switch (see **`-help`** below) in order to see the list of valid ones.

  > **Note**
  >
  > Firebird 2.5 onwards.

- **`-FE[TCH_PASSWORD] <password file name> | stdin | /dev/tty`**

  This switch causes the password for the appropriate user to be read from a file as opposed to being specified on the command line. The file name supplied is *not* in quotes and must be readable by the user running gbak. If the file name is specified as `stdin`, then the user will be prompted for a password. On POSIX systems, the file name `/dev/tty` will also result in a prompt for the password.

  > **Note**
  >
  > Firebird 2.5 onwards.

- **`-M[ETA_DATA]`**

  This switch causes your data to be ignored and not backed up or restored. In a backup, only the database meta data are backed up. When used in a restore, any data in the dump file will not be restored. This switch can be used when creating database clones which are required to contain only the tables, indices etc but none of the data.

- **`-PAS[SWORD] <password>`**

  Supplies the password for the username specified above. This need not be supplied if `ISC_PASSWORD` environment variable exists and has the correct value.

- **`-RO[LE] <SQL role name>`**

  Allows the specification of the role to be used by the connecting user. Not of much practical use and is not normally used in practice.

- **`-U[SER] <username>`**

  Allows the username of the SYSDBA or database owner user to be specified if the database is to be backed up, or, in the case of a restore (with the **`-c[reate]`** switch specified), any valid username can be specified. This need not be supplied if `ISC_USER` environment variable exists and has a correct value for the username.

  Databases can only be backed up by SYSDBA or the database owner. A restore can also be carried out by SYSDBA or the database owner, however, if the **`-c(reate)`** switch is used, *any* validated username can create a new database from a backup file.

- **`-V[ERIFY]`**

  Normally gbak operates quietly with no information written to the display. This switch reverses that situation and causes lots of information to be displayed. The default is to display the output to the screen, but you can redirect the output to a log file using the **`-y`** switch.

- **`-Y <filename> or the text "suppress"`**

  Used in conjunction with the **`-v[erify]`** switch to redirect status messages to a file or device, rather than the screen, or to suppress them altogether.

  If **`-y suppress`** is used, then no information will be written to screen regardless of whether **`-v[erify]`** is specified.

  If a filename is given *and* the **`-v[erify]`** switch is specified, the file will be written to record progress, errors etc.

  > **Warning**
  >
  > All known documentation on this switch mentions that the text should be **`"suppress_output"`** rather than **`"suppress"`**. This is an error in the documentation as the source code for gbak shows that the switch must be **`"suppress`**".

Warning

The log file must not exist. If it does, the backup or recovery operation will fail:

```
tux> rm employee.log
tux> gbak -backup employee.fdb employee.fbk -y employee.log -v

tux> ls -l employee.log
-rw-r--r-- 1 firebird firebird 21610 2010-08-04 10:22 employee.log

tux> gbak -backup employee.fdb employee.fbk -y employee.log -v
gbak:cannot open status and error output file employee.log
```

- **-Z**

  This option displays some information about the version of gbak being used, and optionally, a database. To obtain the version of gbak only, run the command as follows:

```
tux> gbak -z

gbak:gbak version LI-V2.5.0.26074 Firebird 2.5
gbak: ERROR:requires both input and output filenames
gbak:Exiting before completion due to errors

tux> echo $?
1
```

  This displays the current version of gbak, and after displaying a couple of error messages, exits with an error code of 1 as shown above by the **echo** command. This method does not attempt to backup any databases and does not require a username and password to be defined or supplied.

  If you wish to display the version number of the gbak utility and also details of database, you must supply a valid database name *and* backup filename, as follows:

```
tux> gbak -z employee employee.fbk -user sysdba -password secret

gbak:gbak version LI-V2.1.3.18185 Firebird 2.1
gbak:    Version(s) for database employee
Firebird/linux (access method),version LI-V2.1.3.18185 Firebird 2.1
Firebird/linux (remote server),version LI-V2.1.3.18185
Firebird 2.1/tcp (tux)/P11
Firebird/linux (remote interface), version LI-V2.1.3.18185
Firebird 2.1/tcp (tux)/P11
on disk structure version 11.1

tux> echo $?
0
```

  You will note from the above that a valid username and password must be defined on the command line, or by the use of environment variables `ISC_USER` and `ISC_PASSWORD`. This version of the command will exit with a error code of zero.

Warning

This method of calling gbak *will* make a backup of the database. If your database is large, this can take some time to complete and the backup file specified *will* be overwritten if it already exists. Beware.

Note

The output above has been slightly abbreviated to allow it to fit the page width for a pdf.

- **`-help`**

  Help is actually not a valid option, but can be used to display the following screen of information as output from gbak in Firebird 2.0:

```
gbak:legal switches are:
        -B[ACKUP_DATABASE]    backup database to file
        -BU[FFERS]            override page buffers default
        -C[REATE_DATABASE]    create database from backup file
        -CO[NVERT]            backup external files as tables
        -E[XPAND]             no data compression
        -FA[CTOR]             blocking factor
        -G[ARBAGE_COLLECT)    inhibit garbage collection
        -I[NACTIVE]           deactivate indexes during restore
        -IG[NORE]             ignore bad checksums
        -K[ILL]               restore without creating shadows
        -L[IMBO]              ignore transactions in limbo
        -M[ETA_DATA]          backup metadata only
        -MO[DE] <access>      "read_only" or "read_write" access
        -N[O_VALIDITY]        do not restore database validity conditions
        -NOD[BTRIGGERS]       do not run database triggers
        -NT                   Non-Transportable backup file format
        -O[NE_AT_A_TIME]      restore one table at a time
        -OL[D_DESCRIPTIONS]   save old style metadata descriptions
        -P[AGE_SIZE]          override default page size
        -PAS[SWORD]           Firebird password
        -R[ECREATE_DATABASE]  [O[VERWRITE]] create (replace if O[VERWRITE] used)
                                database from backup file
        -REP[LACE_DATABASE]   replace database from backup file
        -RO[LE]               Firebird SQL role
        -SE[RVICE]            use services manager
        -T[RANSPORTABLE]      transportable backup -- data in XDR format
        -USE_[ALL_SPACE]      do not reserve space for record versions
        -USER                 Firebird user name
        -V[ERIFY]             report each action taken
        -Y  <path>            redirect/suppress status message output
        -Z                    print version number
```

Note

The explanation of the **`-m[eta_data]`** switch implies that it is useful in a backup situation only. This is not the case as it can also be used on a restore.

From Firebird 2.5 onwards, there is a new **`-?`** switch to display the list of valid options. The output has slightly different layout and a couple of new options have been added:

```
gbak:legal switches are:
        -B(ACKUP_DATABASE)    backup database to file
        -C(REATE_DATABASE)    create database from backup file (restore)
        -R(ECREATE_DATABASE) [O(VERWRITE)] create (or replace if OVERWRITE used)
                                database from backup file (restore)
        -REP(LACE_DATABASE)   replace database from backup file (restore)

gbak:backup options are:
        -CO(NVERT)            backup external files as tables
        -E(XPAND)             no data compression
        -FA(CTOR)             blocking factor
        -G(ARBAGE_COLLECT)    inhibit garbage collection
        -IG(NORE)             ignore bad checksums
        -L(IMBO)              ignore transactions in limbo
        -NOD(BTRIGGERS)       do not run database triggers
        -NT                   Non-Transportable backup file format
        -OL(D_DESCRIPTIONS)   save old style metadata descriptions
        -T(RANSPORTABLE)      transportable backup -- data in XDR format

gbak:restore options are:
        -BU(FFERS)            override page buffers default
        -FIX_FSS_D(ATA)       fix malformed UNICODE_FSS data
        -FIX_FSS_M(ETADATA)   fix malformed UNICODE_FSS metadata
        -I(NACTIVE)           deactivate indexes during restore
        -K(ILL)               restore without creating shadows
        -MO(DE) <access>      "read_only" or "read_write" access
        -N(O_VALIDITY)        do not restore database validity conditions
        -O(NE_AT_A_TIME)      restore one table at a time
        -P(AGE_SIZE)          override default page size
        -USE_(ALL_SPACE)      do not reserve space for record versions

gbak:general options are:
        -FE(TCH_PASSWORD)     fetch password from file
        -M(ETA_DATA)          backup or restore metadata only
        -PAS(SWORD)           Firebird password
        -RO(LE)               Firebird SQL role
        -SE(RVICE)            use services manager
        -USER                 Firebird user name
        -V(ERIFY)             report each action taken
        -Y  <path>            redirect/suppress status message output
        -Z                    print version number
```

The parentheses shown in the above indicates how much of the switch name you need to use in order to avoid ambiguity. Once you have specified the absolute minimum - the part before the opening '[' - you can use as much of what follows as you wish. For example, to use the **`-b[ackup_database]`** switch the minimum you must supply on the command line is **`-b`** but anything between **`-b`** and **`-backup_database`** will be accepted.

Using the **`-help`** switch like this, or any other invalid switch, will cause gbak to exit with an error code of 1 on Linux and Windows.

Backup Switches

Note

When running gbak, if the *first* filename is a database name, or database alias then gbak will default to taking a backup of the database in transportable format. The backup file will be named as per the second file name supplied on the command line.

Note

You may, if you wish, send the output to standard output rather than a backup file. In this case, you must specify stdout as the dump file name. This is not really of much use, unless you wish to pipe the dump through a tool to modify it in some way. You can pipe the output directly to a gbak restore operation to clone a database without needing an intermediate dump file. An example is given later in this manual.

When carrying out a backup of a database, the following switches, in addition to the common ones above, will be of use:

- **`-B[ACKUP_DATABASE]`**

  This switch is used whenever you wish to take a backup of a database.

- **`-CO[NVERT]`**

  This switch causes any tables, defined as external, to be backed up as if they were normal (non-external) tables. When this dump file is used to restore a database, the tables that were external in the original database will no longer be external.

- **`-E[XPAND]`**

  Normally, gbak will compress the output file. This switch prevents that compression from taking place.

- **`-FA[CTOR] <block size>`**

  If backing up to a physical tape device, this switch lets you specify the tape's blocking factor.

- **`-G[ARBAGE_COLLECT]`**

  The use of this switch prevents Firebird's garbage collection from taking place while gbak is running. Normally gbak connects to the database as any other connection would and garbage collection runs normally. Using this switch prevents garbage collection from running during the course of the backup. This can help speed up the backup.

- **`-IG[NORE]`**

  This switch causes gbak to ignore bad checksums in the database. This can be used to attempt to backup a database that failed due to checksum errors. There is no guarantee that the data will be usable though, so it is best to take other precautions to preserve your data.

- **`-L[IMBO]`**

If you have a two-phase transaction (across two different databases) that failed because a server died before the commit, but after the changes were made, you have a limbo transaction. This switch forces the backup to ignore data from such broken transactions. It should not be used for normal backups and only used, like the **`-ig[nore]`** switch to attempt to recover from a failure.

- **`-M[ETA_DATA]`**

  See above.

- **`-NT`**

  This switch turns off the **`-t[ransportable]`** switch (which is on by default) and causes the dump file to be created using platform dependent formats. If you use this switch to create a backup then you can only restore the backup on a similar platform. You cannot, for example, take a dump file from Linux over to a Windows server.

- **`-OL[D_DESCRIPTIONS]`**

  This switch is unlikely to be used. It has been deprecated. Its purpose is to force the backup to be made using the older InterBase versions' format of meta data descriptions.

- **`-PAS[SWORD] <password>`**

  See above.

- **`-RO[LE] <role name>`**

  See above.

- **`-SE[RVICE] <servicename>`**

  This switch causes gbak to backup a remote database via the service manager. This causes the backup file to be created on the remote server, so the path format and filename must be valid on the remote server. The servicename is currently always the text **`service_mgr`**.

  > **Note**
  >
  > You can use this option to connect to a locally hosted database as well. If you do, taking a backup using this option can run quicker than accessing the database directly. See the section below on speeding up backups.

- **`-T[RANSPORTABLE]`**

  The default dump file format is transportable. Transportable backup files are written in a format known as *external data representation* (XDR) format and it is this format which allows a dump taken on a server of one type to be successfully restored on a server of another type.

- **`-USER <username>`**

  See above.

- **`-V[ERIFY]`**

  See above.

- **`-Y <full name of log file> or the text "suppress"`**

  See above.

Restore Switches

> **Note**
>
> When running a gbak command, if the *first* filename is a database backup file name then gbak will default to running a recovery of the database provided that you specify one of **`-c[create database]`**, **`-rep[lace_database]`** or **`-r[ecreate_database]`** in order to make your intentions clear. The database will be restored to whatever file name is specified as the second file name on the command line..

> **Note**
>
> You may, if you wish, read the dump data directly from standard input rather than a backup file. In this case, you must specify stdin as the dump file name. You could pipe a gbak dump operation directly to a gbak restore operation to clone a database without needing an intermediate dump file. An example is given later in this manual.

When carrying out a restore or replacement of a database, the following switches, in addition to the common ones above, will be of use:

- **`-BU[FFERS] <number of buffers>`**

  This switch sets the default database cache size (in number of database pages) for the database being restored. If a database is being overwritten then this setting will overwrite the previous setting for the cache size.

- **`-C[REATE_DATABASE]`**

  This switch causes a new database to be created from the backup file. The database file must not exist or the restore will fail. Either this switch or **`-rep[lace_database]`** or **`-r[ecreate_database]`** must be specified.

- **`-FIX_FSS_D[ATA]`**

  This switch forces gbak to fix malformed UNICODE_FSS character data during a restore.

  This switch, and the following one, should not be required under normal circumstances. However, if a restore operation fails with a "malformed string" error, the message output from gbak will refer the user to one or both of these switches to fix the malformed UNICODE_FSS data or meta data as appropriate.

  > **Note**
  >
  > Firebird 2.5 onwards.

- **`-FIX_FSS_M[ETADATA]`**

  This switch forces gbak to fix malformed UNICODE_FSS metadata during a restore.

  This switch, and the preceding one, should not be required under normal circumstances. However, if a restore operation fails with a "malformed string" error, the message output from gbak will refer the user to one or both of these switches to fix the malformed UNICODE_FSS data or meta data as appropriate.

> **Note**
>
> Firebird 2.5 onwards.

- **`-I[NACTIVE]`**

  This switch can be used to restore a database when a previous restore attempt failed due to index errors. All indexes in the restored database will be inactive.

- **`-K[ILL]`**

  This switch restores the database but doesn't recreate any shadow files that existed previously.

- **`-M[ETA_DATA]`**

  See above.

- **`-MO[DE] <access>`**

  This switch allows the database being restored to be set to the given access mode when opened. By default, the mode is taken from the database that was dumped.

- **`-N[O_VALIDITY]`**

  This switch is similar to the **`-i[nactive]`** switch above, except, it disabled all *check* constraints in the restored database.

- **`-NOD[BTRIGGERS]`**

  New switch from Firebird 2.1 which prevents the *database triggers* from firing on a restore. Database triggers are a new feature in Firebird 2.0 onwards and are different from *table triggers* which will continue to fire.

- **`-O[NE_AT_A_TIME]`**

  This switch causes the restore to restore one table at a time. This can be useful when a previous restore failed due to data errors. Normally, a restore takes place in a single transaction with a single commit at the end of the restore. If the restore is interrupted for any reason, an empty database is the end result. Using the **`-o[ne_at_a_time]`** option uses a transaction for each table and commits after each table has been restored.

- **`-P[AGE_SIZE] <new page size>`**

  Use this switch to change the default database page size. By default, the database is restored using a page size the same as the one that was in use when the database was dumped.

- **`-PAS[SWORD] <password>`**

  See above.

- **`-R[ECREATE_DATABASE] [O[VERWRITE]]`**

  New from Firebird 2.x. Create (or replace if **`o[verwrite]`** is used) the named database from the backup file. The database filename should not already exist or an error will occur. This is not the case if the **`o[verwrite]`** option is also used.

This is a new switch and is deliberately abbreviated to **`-r`** to try to prevent unsuspecting DBAs from overwriting an existing database thinking that the **`-r`** was abbreviated from **`-restore`**. Now, it takes special effort to manage this especially as **`-restore`** was never actually a valid switch; **`-r`** was in fact an abbreviation of **`-replace_database`** and it did this by *first* deleting the existing database file and *then* recreating it from the backup.

Using **`-r[ecreate_database] o[verwrite]`** is effectively the same as using **`-rep[lace_database]`**.

- **`-REP[LACE_DATABASE]`**

  Replace database from backup file. This switch used to be abbreviated to **`-r`** in previous (to Firebird 2.x) versions. This switch will be removed in a version of Firebird later than 2.1.3 (where it still exists). You are advised to use the **`-r[ecreate_database] o[verwrite]`** switch instead.

- **`-SE[RVICE] <servicename>`**

  Use the services manager on a remote database to restore a remote database. The servicename is currently always the text **`service_mgr`**.

  > **Note**
  >
  > You can use this option to connect to a locally hosted database as well. If you do, restoring a backup using this option can run quicker than accessing the database directly. See the section below on speeding up restores.

- **`-USE_[ALL_SPACE]`**

  This switch forces the restore to use 100% of each database page and thus not leave any room for changes. If you omit this switch, some space will be kept free for subsequent changes. Using this switch is likely to be only of practical use where the database is created and used in read-only mode and no updates to existing data are required.

  > **Warning**
  >
  > Once a database has been restored with this option specified, *all* database pages will be filled to 100% and no free space will be left for updates. Use of this switch set a flag in the database header page to signal that *all* pages are to be filled to 100% - this applies to any new pages created after the restore.
  >
  > You can override this setting, using **gfix -use full | reserve database_name** where **`full`** uses 100% of each page and **`reserve`** reserves some space for subsequent updates. See the gfix manual for more details.

- **`-USER <username>`**

  See above.

- **`-V[ERIFY]`**

  See above.

- **`-Y <name of log file>`**

  See above.

Backup Mode

Before you consider using other tools to take a backup of your Firebird database, make sure that you know what the tools do and how a running database will be affected by them. For example, if you use Winzip to create a compressed copy of a database and you do it when users are accessing the system, the chances of a successful restore of that database are slim. You must either always use the gbak or nbackup tools which know how the database works, or, use gfix to shut the database down completely before you even attempt to backup the database file(s).

Gbak creates a consistent backup of the database by starting a transaction that spans the backup period. When the backup is complete, the transaction is ended and this means that the backup process can be run while users are working in the database. However, any transactions started after the backup process begins will not have any of the changed data written to the backup file. The backup will represent a copy of the entire database at the moment the backup began.

The dump file created by a default gbak backup is cross platform (transportable) so a backup taken on a Windows server can be used to recreate the same database on a Linux server, or on any other platform supported by Firebird. This is not true of the copies of your database taken (while the database was closed!) with tools such as Winzip etc. Those copies should only ever be used to restore a database on the same platform as the one copied.

Important

Always backup the database with the version of gbak supplied with the running database server.

And one final thought on backups, regardless of the fact that the backup finished with no errors, exited with an error code of zero and all appears to be well, how do you actually know that the backup file created is usable? The short answer is, you don't. Whenever you have a valuable database - and they all should be - you are strongly advised to take your backup files and use them to create a test restore of a database either on the same server or even better, on a different one. Only by doing this can you be certain of a successful backup.

The following example shows a backup being taken on a server named *linux* and used to create a clone of the database on another Linux server named *tux* to make sure that all was well. First of all, the backup on *linux*:

```
linux> gbak -backup -verify -y backup.log employee employee.fbk
linux> gzip -9 employee.fbk
```

Note

Note that the above gbak command can be written as follows, leaving out the **`-b[ackup]`** switch as gbak defaults to running a backup when no other suitable switches are specified:

```
linux> gbak -verify -y backup.log employee employee.fbk
```

Then, on the *tux* server:

```
tux> scp norman@linux:employee.fbk.gz ./

Using keyboard-interactive authentication.
Password:
employee.fbk.gz            |         19 kB |  19.3 kB/s | ETA: 00:00:00 | 100%
```

```
tux> gunzip employee.fbk.gz
tux> gbak -replace -verify -y restore.log employee.fbk employee.restore.test
```

At this point, the restore has worked and has overwritten the previous database known as employee.restore.test.

The actual location of the database for the database employee.restore.test is defined in the `aliases.conf` file in `/opt/firebird` on the server. In this test, it resolves to `/opt/firebird/databases/employee.restore.fdb`.

For further proof of reliability, the application may be tested against this clone of the live database to ensure all is well.

Speeding up the Backup

There are a couple of tricks you can use to speed up the backup. The first is to prevent the garbage collection from being carried out while the backup is running. Garbage collection clears out old record versions that are no longer required and this is usually covered by a sweep - manual or automatic - or by a full table scan of any affected table. As gbak accesses all the rows in the tables being backed up, it too will trigger the garbage collection and, if there have been a large number of updates, can slow down the backup. To prevent garbage collection during the backup, use the **`-g[arbage_collect]`** option.

```
tux> gbak -backup -garbage_collect employee /backups/employee.fbk
```

The second option is to backup the database using the **`-se[rvice]`** option. Although this is used to perform remote backups, it can be used locally as well. Using this option can help speed up your backups. It simply avoids the data being copied over the TCP network which can slow down the actions of the backup.

```
tux> gbak -backup -service tux:service_mgr employee /backups/employee.fbk
```

The example above backs up the employee database, on the tux server, "remotely" using the service manager. The tux server is, of course, where the command is running, so it isn't really running remotely at all.

You can, of course, combine the **`-g[arbage_collect]`** and **`-se[rvice]`** options.

Restore Mode

Backups taken with the gbak application from one version of Firebird - or InterBase - can be used by later versions of Firebird to restore the database, however, while this may result in an upgrade to the On Disc Structure (ODS) for the database in question, the SQL Dialect will never be changed. If you backup an InterBase dialect 1 database and then use the dump file to recreate it under Firebird 2.1, for example, the ODS will be updated to 11.1 but the SQL Dialect will remain as 1.

Important

Always restore the database with the version of gbak supplied with the database server you wish to run the (new) database under. However, gbak from Firebird 2.1 can be used to restore a database onto any version of Firebird.

You can, if you wish, change the SQL Dialect using gfix.

Under normal circumstances, restoring a database takes place as a single transaction. If the restore is successful, a commit at the end makes the data permanent, if not, the database will be empty at the end.

The restore option **`-o[ne_at_a_time]`** will use a transaction for each table and if the table is restored with no errors, a commit is executed rendering that table permanent regardless of what happens with subsequent tables.

Restore Or Recreate?

Should a database be restored or replaced? Restoring a database is the process by which you take the existing file and delete prior to replacing it on disc with a backup copy. Gbak does this when you specify the **`-r[ecreate_database] o[verwrite]`** switch or the **`-rep[lace_database]`** switch. What is the difference?

If a database exists on disc and you ask gbak to restore it using one of the two switches above, you might corrupt the database especially if the database is in use and has not been shut down using gfix. In addition, if you have only partially completed the restore of a database, and some users decide to see if they can login, the database may well be corrupted.

Finally, if the restore process discovers that the dump file is corrupt, the restore will fail and your previously working database will be gone forever.

It can be seen that restoring a database can be a fraught experience.

For security, always recreate the database with a new name - a clone - and update the `aliases.conf` to reflect the new name. This way, your users will always refer to the database by the alias regardless of the actual filename on the server.

Malformed String Errors During Restores

During a restore operation, most likely when restoring a backup taken using an older gbak version, it is possible to see failure messages, in gbak's output, indicating malformed Unicode strings. The reason that these may occur is as explained by Helen Borrie:

> The source text of stored procedures (and several other types of object, such as CHECK constraints) is stored in a blob, as is the "compiled" BLR code. When you restore a database, the BLR is not recreated: the same BLR is used until next time you recreate or alter the object.
>
> Historically, the engine did not do the right thing regarding the transliteration of strings embedded in the source and the BLR. In v.2.1 and 2.5 a lot of work was done to address the international language issues, as you probably know. A side effect of this was that everything that was read from data and meta data became subject to "well-formedness" checks. Hence, on restoring, those previously stored source and BLR objects are throwing "malformed string" errors when gbak tries to read and write the data in these system table records. This very old bug affects user blobs, too, if they have been stored using character set NONE and the client is configured to read a specified character set to which the stored data could not be transliterated.
>
> In v.2.1 there were scripts in `../misc` that you could run to repair the meta data blobs and also use as a template for repairing the similar errors in blobs in your user data. The repair switches were added to the gbak restore code in v.2.5 to do the same corrections to meta data and data, respectively, during the process of restoring a database for upgrade.

Speeding up the Restore

The restoration of a database, from a backup, can be made to execute quicker if the **`-se[rvice]`** option is used. Although this is used to perform remote restores, it can be used locally as well. It simply avoids the data being copied over the TCP network which can slow down the actions of the restore.

```
tux> gbak -replace -service tux:service_mgr /backups/employee.fbk employee
```

The example above backs up the employee database, on the tux server, "remotely" using the service manager. The tux server is, of course, where the command is running, so it isn't really running remotely at all.

You can, of course, combine the **`-g[arbage_collect]`** and **`-se[rvice]`** options.

Security Of Backups

As you have seen above anyone, with a valid username and password, can restore a gbak database dump file provided that they are not overwriting an existing database. This means that your precious data can be stolen and used by nefarious characters on their own servers, to create a copy of your database and see what your sales figures, for example, are like.

To try and prevent this from happening, you are advised to take precautions. You should also try and prevent backups from being accidentally overwritten before they have expired. Some precautions you can take are:

- Always set the dump file to be read-only after the backup is complete. This helps prevent the file from being overwritten.
- Alternatively, incorporate the date (and time?) in your backup filenames.
- Keep backups in a safe location on the server. Storing backups in a location with restricted access helps reduce the chances of your backup files 'escaping' into the wild.
- Keep tape copies of your backups very secure. A locked safe or off-site location with good security is advisable. The off-site location will also be of use after a total disaster as the backups are stored in a separate location to the server they are required on.
- Backup to a partition or disc that has encryption enabled.
- Make sure that only authorised staff have access to areas where backups are kept.
- Always test your backups by cloning a database from a recent backup.

In Firebird 2.1, there is an additional security feature built into gbak and all the other command-line utilities. This new feature automatically hides the password if it is supplied on the command line using the -password switch. Gbak replaces the password with spaces - one for each character in the password. This prevents other users on the system, who could run the **ps** command and view your command line and parameters, from viewing any supplied password. In this manner, unauthorised users are unable to obtain the supplied password.

```
tux> gbak -b -user SYSDBA -passw secret employee /backups/employee.fbk
```

```
tux> ps efx| grep -i gba[k]
20724 ... gbak -backup -user SYSDBA -passw           employee employee.fbk
... (lots more data here)
```

You can see from the above that the password doesn't show up under Firebird 2.1 as each character is replaced by a single space. This does mean that it is possible for someone to work out how *long* the password *could* be and that might be enough of a clue to a dedicated cracker. Knowing the length of the required password does make things a little easier, so for best results use a random number of spaces between **`-passw`** and the actual password. The more difficult you make things for the bad people on your network, the better.

Backup & Restore Recipes

The following recipes show examples of backup and restore tasks using gbak. These are probably the commonest cases that you will encounter as a DBA. All the examples use the employee database supplied with Firebird and the actual location is correctly configured in `aliases.conf`. Each of the following recipes is run with the assumption that the environment variables `ISC_USER` and `ISC_PASSWORD` have been given suitable values.

Backup & Restore Prerequisites

If you replace an open and running database, there is a good chance that you will corrupt it. For best results and minimal chance of corrupting a database, you should close it before replacing it. To close a database, use gfix as follows:

```
tux> gfix -shut -tran 60 employee
```

The example above prevents any new transaction from being started which prevents new queries being executed or new sessions connecting to the database. It will wait for up to 60 seconds for everyone to logout and for all current transactions to complete before shutting down the database. If any long running transactions have not completed by the end of 60 seconds, the shutdown will timeout and the database will remain open.

> **Note**
>
> After the restore of the database has completed, the database will automatically be opened again for use.

A Simple Backup & Restore

This example takes a backup, then immediately overwrites the original database using the new backup. This is not normally a good idea as the first action of a restore is to wipe out the database.

```
tux> # Backup the database.
tux> gbak -backup employee /backups/employee.fbk

tux> # Restore the database.
tux> gfix -shut -tran 60 employee
tux> gbak -replace /backups/employee.fbk employee
```

Meta Data Only

It is possible to use gbak to recreate an empty database containing only the various domains, tables, indices and so on, of the original database but none of the data. This can be useful when you have finished testing your application in a test environment and wish to migrate the system to a production environment, for example, but starting afresh with none of your test data.

```
tux> #Backup only the database metadata.
tux> gfix -shut -tran 60 employee
tux> gbak -backup -meta_data employee employee.meta.fbk
```

When the above dump file is restored on the production server, only the meta data will be present.

There is another way to create a database with no data and only the meta data. Simply restore from an existing dump which contains the data and supply the **`-m[eta_data]`** switch to the restore command line. The database will be restored but none of the original data will be present.

```
tux> #Restore only the database metadata.
tux> gbak -create employee.fbk mytest.fdb -meta_data
```

The **`-m[eta_data]`** switch can be used on either a backup or a restore to facilitate the creation of a clone database (or overwrite an existing one) with no actual data.

Splitting The Backup

The gsplit filter application, documented in its own manual, doesn't actually work anymore. This filter was supplied with old versions of InterBase and Firebird to allow large database backups to be split over a number of files so that file system limits could be met. Such limits could be the size of a CD, the 2GB limit on individual file sizes on a DVD, where some Unix file systems have a 2 GB limit and so on.

Gbak allows the dump files to be split into various sizes (with a minimum of 2048 bytes) and will only create files it needs.

```
tux> # Backup the database to multiple files.
tux> gbak -backup employee /backups/emp.a.fbk 600m /backups/emp.b.fbk 600m
```

The sizes after each filename indicate how large that particular file is allowed to be. The default size is bytes, but you can specify a suffix of **`k`**, **`m`** or **`g`** to use units of kilo, mega or gigabytes.

If the dump completes before writing to some files, then those files are not created. A dump file is only ever created when it must be.

The size of the final dump file will be quietly ignored if the database has grown too large to allow a truncated backup to complete. If, in the example above, the backup needs a total of 1500M, then the last file would be written to a final size of 900m rather than the 600m specified.

To restore such a multi-file backup requires that you specify all of the filenames in the dump and in *the correct order*. The following example shows the employee database above being restored from the two files dumped above:

```
tux> # Restore the database from multiple files.
tux> gfix -shut -tran 60 employee
tux> gbak -replace /backups/employee.a.fbk /backups/employee.b.fbk employee
```

Change The ODS

Normally the ODS used is the one in force by the version of Firebird used to restore the database. So, the examples above will actually change the ODS when the database is restored. The backup should be taken using the gbak utility supplied by the old ODS version of InterBase or Firebird. The restore should be carried out using gbak from the newer version of Firebird.

```
tux> setenv_firebird 2.0
Firebird environment set for version 2.0.

tux> # Check current ODS version (as root user!)
tux> gstat -h employee|grep ODS
        ODS version             11.0

tux> # Backup the (old) database.
tux> gbak -backup employee /backups/employee.2_0.fbk

tux> setenv_firebird 2.1
Firebird environment set for version 2.1.

tux> # Recreate the database and upgrade the ODS.
tux> gfix -shut -tran 60 employee
tux> gbak -replace /backups/employee.2_0.fbk employee

tux> # Check new ODS version (as root user!)
tux> gstat -h employee|grep ODS
        ODS version             11.1
```

After the above, the old 2.0 Firebird database will have been recreated - wiping out the old database - as a Firebird 2.1 database with the corresponding upgrade to the ODS from 11.0 to 11.1.

The script `setenv_firebird` is not supplied with Firebird and simply sets `PATH` etc to use the correct version of Firebird as per the supplied parameter.

Change The Cache Size

The default database cache is created when the database is created, or subsequently by using gfix. Gbak can restore a database and reset the default cache size as well. The process is as follows:

```
tux> # Check current cache size (as root user!)
tux> gstat -h employee | grep -i buffer
        Page buffers            0

tux> # Restore the database & change the cache size.
tux> gfix -shut -tran 60 employee
tux> gbak -replace -buffer 200 /backups/employee.fbk employee

tux> # Check the new cache size (as root user!)
tux> gstat -h employee | grep -i buffer
        Page buffers            200
```

The default cache size is used when the number of buffers is zero, as in the first example above. Gbak allows this to be changed if desired. Gbak, however, cannot set the cache size back to zero. You must use gfix to do this.

Change The Page Size

Similar to the example above to change the default database cache size, the database page size can also be changed using gbak.

```
tux> # Check current page size (as root user!)
tux> gstat -h employee | grep -i "page size"
        Page size               4096

tux> # Restore the database & change the page size.
tux> gfix -shut -tran 60 employee
tux> gbak -replace -page_size 8192 /backups/employee.fbk employee

tux> # Check the new page size (as root user!)
tux> gstat -h employee | grep -i "page size"
        Page size               8192
```

Create A Read-Only Database Clone

Sometimes you do not want your reporting staff running intensive queries against your production database. To this end, you can quite easily create a clone of your production database on a daily basis, and make it read-only. This allows the reporting team to run as many intensive reports as they wish with no ill effects on the production database and it prevents them from inadvertently making changes.

The following example shows the production employee database running on Linux server *tux*, being cloned to the reporting team's Linux server named *tuxrep*. First on the production *tux* server:

```
tux> # Backup the production database.
tux> gbak -backup employee /backups/employee.fbk
```

Then on the reporting team's *tuxrep* server:

```
tuxrep> # Scp the dump file from tux.
tuxrep> scp fbuser@tux:/backups/employee.fbk ./
Using keyboard-interactive authentication.
Password:
employee.fbk              |         19 kB |  19.3 kB/s | ETA: 00:00:00 | 100%

tuxrep> # Restore the employee database as read-only.
tuxrep> gfix -shut -tran 60 employee
tuxrep> gbak -replace -mode read_only employee.fbk employee

tuxrep> # Check database mode (as root user)
tuxrep> gstat -h employee|grep -i attributes
        Attributes              no reserve, read only
```

Create a Database Clone Without a Dump File.

You may use gbak to create a clone of a database, on the same server, without needing to create a potentially large dump file. To do this, you pipe the output of a gbak backup directly to the input of a gbak restore, as follows.

```
tux> # Clone a test database to the same server, without requiring a dump file.
tux> gbak -backup emptest stdout | gbak -replace stdin emptest_2
```

You will notice that the output file name for the backup is `stdout` and the input file name for the restore is `stdin`. This ability to pipe standard output of one process to the standard input of another, is how you can avoid creating an intermediate dump file. The commands above assume that there are suitable alias names set up for both emptest and emptest_2. If not, you will need to supply the full path to the two databases rather than the alias.

The **`-replace`** option on the restore process will overwrite the database name specified - as an alias or as a full path - if it exists and will create it anew if it doesn't. You may also use the **`-recreate overwrite`** option as an alternative. Both have the same result.

If you don't want to overwrite any existing databases, use **`-create`** which will only create a database if it doesn't already exist, and will exit with an error if it does. In POSIX compatible systems, the error code in `$?` is 1 in this case.

Further examples of backing up and restoring remote databases over ssh, using the stdin and stdout file names, can be seen below.

Backup & Restore With & Without Shadow Files.

Databases can have shadow files attached in normal use. Gbak happily backs up and restores those as well and in normal use, shadow files will be recreated. Should you wish to restore the database only and ignore the shadows, gbak can do that for you as the following example shows.

```
tux> # Check current shadows, use isql as gstat is broken.
tux> isql employee

Database:  employee
SQL> show database;
Database: employee
        Owner: SYSDBA
 Shadow 1: "/opt/firebird/shadows/employee.shd1" manual
 Shadow 2: "/opt/firebird/shadows/employee.shd2" manual
...

SQL> quit;

tux> # Restore the database preserving shadow files.
tux> gfix -shut -tran 60 employee
tux> gbak -replace overwrite /backups/employee.fbk employee

tux> # Check shadows again, use isql as gstat is broken.
tux> isql employee

Database:  employee
SQL> show database;
Database: employee
        Owner: SYSDBA
 Shadow 1: "/opt/firebird/shadows/employee.shd1" manual
 Shadow 2: "/opt/firebird/shadows/employee.shd2" manual
...

SQL> quit;


tux> # Restore the database killing shadow files.
tux> gfix -shut -tran 60 employee
tux> gbak -replace overwrite -kill /backups/employee.fbk employee

tux> # Check shadows again, use isql as gstat is broken.
tux> isql employee
```

```
Database:  employee
SQL> show database;
Database: employee
        Owner: SYSDBA
...

SQL> quit;
```

> **Note**
>
> I use isql in the above examples as **`gstat -h`** seems to get confused about how many shadows there are on a database. It reports zero when there are two, eventually it catches up and reports that there are two, then, if you kill a shadow, it reports that there are now three!

Remote Backups & Restores

Firebird's gbak utility can make backups of a remote database. To do this, you need to connect to the service manager running on the remote server, this is normally called *service_mgr*. The following example shows the Firebird employee database on server *tuxrep* being backed up from the server *tux*. The backup will be written to the remote server, in other words, the backup file will be created on the *tuxrep* server and not on the *tux* one. The network protocol in use is TCP.

```
tux> # Backup the reporting database on remote server tuxrep.
tux> gbak -backup -service tuxrep:service_mgr employee /backups/remote_backup.fbk
```

The backup file will have the same owner and group as the Firebird database server - on Unix systems at least.

It is also possible to restore a remote database in this manner as well, and gbak allows this.

```
tux> # Restore the read-only reporting database on remote server tuxrep.
tux> gbak -replace -mode read_only -service tuxrep:service_mgr \
            /backups/remote_backup.fbk employee
```

> **Note**
>
> The above example uses the handy Unix ability to split a long line over many shorter ones using a back slash as the *final* character on the line.

As ever, you are advised to beware of replacing a database in case there are problems during the restore. The example above recreates the existing database in read-only mode but this need not always be the case.

A remote backup can also be run on the database server itself! On Windows, this makes no difference, but on Unix systems, this local-remote method of backups and restores reduces network traffic. The 'remote' server, in this case, is not actually remote it is just the method of running the backup - connecting to the service manager - that implies remoteness.

```
tux> # Backup the employee database on this server, but pseudo-remotely!
tux> gbak -backup -service tux:service_mgr employee /backups/remote_backup.fbk
```

And corresponding restores can also be run 'remotely':

```
tux> # Restore the employee database on this server, but pseudo-remotely!
tux> gbak -replace -service tux:service_mgr /backups/remote_backup.fbk employee
```

The format of the parameter used for the -service switch is different according to the nature of the network protocol in use:

- TCP

  When using TCP networks, the parameter separator is a colon, as in **`-service server_name:service_mgr`**.

- Named pipes

  When using named pipes, the parameter requires two leading back slashes and the separator is another back slash, as in **`-service \\server_name\service_mgr`**.

Remote Backups and Restores Using SSH

As shown above, you can use the special file names stdin and stdout to backup and restore a database to a separate database on the same server. However, you can also use the same tools, over an SSH connection to a remote server, and pass the backup of one database directly to a restoration of a separate one.

The first example copies a local database to a remote server where Firebird is running and the firebird user has its environment set up so that the gbak tool is on $PATH by default, on login.

Note

In each of the following examples, the **`-user sysdba`** and **`-password whatever`** parameters on the command lines have been replaced by {...}. When executing these commands, any remote gbak commands will require to have them specified unless the firebird user on the remote database(s) has `ISC_USER` and `ISC_PASSWORD` defined in the `.profile` or `.bashrc` (or equivalent) login files. However, that is a *seriously* bad idea and incredibly insecure.

```
tux> # Clone a test database to a different server, without requiring a dump file.
tux> gbak -backup employee stdout | \
ssh firebird@tuxrep "gbak {...} -replace stdin emptest"
```

When the above is executed, you will be prompted for a password for the remote firebird user on server tuxrep, assuming that you don't have a proper SSH key-pair already set up and active. The command will replace the local database according to the alias name emptest but you can, if required, supply full path names for the databases. The following shows an example of the above being executed.

```
tux> # Clone a test database to a different server, without requiring a dump file.
tux> gbak -backup employee stdout | \
ssh firebird@tuxrep "gbak {...} -replace stdin emptest"

firebird@tuxrep's password:
```

As you can see, there's not much in the way of output, but you can connect remotely and check:

```
tux> isql {...} tuxrep:emptest

Database:  tuxrep:emptest

SQL> show database;

Database: tuxrep:emptest
        Owner: SYSDBA
```

```
PAGE_SIZE 4096
...
```

The next example, shows a remote database being backed up to a local one, in a similar manner.

```
tux> ssh firebird@tuxrep "gbak -backup {...} emptest stdout" | \
gbak -create stdin data/tuxrep_emptest.fdb

firebird@tuxrep's password:

tux> ls data

employee.fdb  tuxrep_emptest.fdb
```

You can see that a new `tuxrep_emptest.fdb` database has been created. Does it work? Checking with isql shows that it does.

```
tux> isql data/tuxrep_emptest.fdb

Database:  data/tuxrep_emptest.fdb

SQL> quit;
```

The final example shows how to backup a remote database on one server, to a remote database on another.

```
tux> ssh firebird@tuxrep "gbak -backup {...} emptest stdout" |  \
ssh firebird@tuxqa "gbak -create {...} stdin data/tuxrep_empqa.fdb"

firebird@tuxrep's password:
firebird@tuxqa's password

tux> ssh firebird@tuxqa "ls data"

employee.fdb  tuxrep_empqa.fdb
```

Using External Tools

Gbak and nbackup are the best tools to use when backing up and/or restoring Firebird databases. They have been extensively tested and know the internals of the database and how it works, so the chances of these tools corrupting your valuable data are very slim. However, some DBAs still like to use external tools (those not supplied with Firebird) to make backups for whatever reason.

Because it is not possible for external tools to know where a database is to be found, given the alias name, the script writer and/or DBA must explicitly find out the correct location of the database file(s) and supply these to the external tool. To make this simpler for script writers, my own installation uses a standard in my `aliases.conf` file as follows:

- The database alias must start in column one.
- There must be a single space before the equals sign (=).
- There must be a single space after the equals sign (=).
- Double quotes around the database filename is not permitted - it doesn't work for the Firebird utilities either.

- Databases are all single file databases.

The last rule applies to my installation only and means that the following simple backup script will work. If multiple file databases were used, more coding would be required to take a backup using external tools.

```
tux> cat /opt/firebird/aliases.conf
# ---------------------------------------------------------
# WARNING: Backup Standards require that:
#          The database name starts in column 1.
#          There is a single space before the equals sign.
#          There is a single space after the equals sign.
#          The path has no double quotes (they don't work!)
# ----------------------------------------------------------
employee = /opt/firebird/examples/empbuild/employee.fdb
```

The following shows the use of the gzip utility on a Linux server to take and compress a backup of a running database. The following is run as the root user due to the requirement to run gfix to shut down the database.

```
tux> # Backup the production employee database using gzip.
tux> gfix -shut -tran 60 employee
tux> DBFILE=`grep -i "^employee =" /opt/firebird/aliases.conf | cut -d" " -f3`
tux> gzip -9 --stdout $DBFILE > /backups/employee.fdb.gz
```

The restore process for this database would be the reverse of the above. Again, the following runs as root.

```
tux> # Restore the production employee database from a gzip backup.
tux> gfix -shut -tran 60 employee
tux> DBFILE=`grep -i "^employee =" /opt/firebird/aliases.conf | cut -d" " -f3`
tux> gunzip --stdout /backups/employee.fdb.gz > $DBFILE

tux> # Make sure firebird can see the file.
tux> chown firebird:firebird $DBFILE
```

Gbak Caveats

The following is a brief list of gotchas and funnies that I have detected in my own use of gbak. Some of these are mentioned above, others may not be. By collecting them all here in one place, you should be able to find out what's happening if you have problems.

Gbak Default Mode

If you do not specify a mode switch such as **`-b[ackup]`** or **`-c[reate]`** etc, then gbak will perform a backup as if the **`-b[ackup]`** switch had been specified - provided that the other switches specified are correct for a backup.

Warning

This detection of whether you are attempting a backup or a restore means that if you use the **`-z`** command line switch to view gbak information, then you *will* create a backup - and overwrite the backup file you supply - if the command line also has a database name and a backup file name present. This assumes that there is a way for gbak to determine the username and password to be used - either as command line parameters or via defined environment variables.

Normal Versus Privileged Users

Only a SYSDBA or the owner of a database can take a backup of the database, however, *any* authenticated user can restore a database backup using the **`-c[reate]`** switch. This means that you must make sure you prevent your backup files from falling into the wrong hands because there is nothing then to stop unauthorised people from seeing your data by the simple process of restoring *your* backups onto *their* server.

The database restore will fail, of course, if the user carrying it out is not the database owner and a database with the same filename already exists.

Silent Running?

The **`-y suppress_output`** switch is supposed to cause all output to be suppressed. Similar in fact to running with **`-v[erify]`** not specified. However, all it seems to do is cause the output (according to the **`-v[erify]`** switch setting) to be written to a file called `suppress_output`, however this only works once because the next run of gbak with **`-y suppress_output`** will fail because the file, `suppress_output`, already exists.

It is possible that this problem was introduced at version 2 for Firebird, because both 2.0 and 2.1 versions actually use the **`-y suppress`** switch rather then **`-y suppress_output`**. Using this (shorter) option does work as intended and the output is indeed suppressed.

Gbak log file Cannot Be Overwritten

If you specify a log file name with the **`-y <log file>`** switch, and the file already exists, then even though the firebird user owns the file, and has write permissions to it, gbak cannot overwrite it. You must always specify the name of a log file that doesn't exist. On Linux systems, the following might help:

```
tux> # Generate unique dump and logfile name.
tux> FILENAME=employee_`date "+%Y%m%d_%H%M%S"`

tux> # Shut down and Backup the database
tux> gfix -shut -tran 60 employee
tux> gbak -backup employee /backups/${FILENAME}.fbk -y /logs/${FILENAME}.log -v
```

The above is quite useful in as much as it prevents you from overwriting previous backups that may be required. The downside is that you now need to introduce a housekeeping system to tidy away old, unwanted backups to prevent your backup area filling up.

Use of 'stdin' or 'stdout' File Names

Gbak recognizes the literal strings 'stdin' and 'stdout' as source or destination filenames. In POSIX systems, when the standard input and/or standard output channels are used, it is not permitted to execute seek operations on these channels. Using 'stdin' or 'stdout' as file names with gbak will force gbak to use processing that will not seek on the input or output channels, making them suitable for use in pipes - as per the examples in the recipes section above.

These file names, while they appear to be POSIX names, are definitely not synonyms for `/dev/stdin` or `/dev/stdout`, they are simply literals that gbak checks for while processing its parameters. Do not attempt to use names `/dev/stdin` or `/dev/stdout` in a piped process as it will most likely fail.

If you wish to create a dump file actually named either stdin or stdout, then you should specify the filename as a full, or relative, path name such as `./stdin` or `./stdout`, which causes gbak to treat them as a literal file name rather than a special file name that causes different to normal processing.during the dump or restore process.

Appendix A: Document history

The exact file history is recorded in the `manual` module in our CVS tree; see http://sourceforge.net/cvs/?group_id=9028 . The full URL of the CVS log for this file can be found at http://firebird.cvs.sourceforge.net/viewvc/firebird/ manual/src/docs/firebirddocs/fbutil_gbak.xml?view=log

Revision History

1.0	10 October 2009	ND	Created as a chapter in the Command Line Utilities manual.
1.1	20 October 2009	ND	More minor updates and converted to a stand alone manual.
1.2	24 November 2009	ND	Corrected the section on **`-y Suppress_output`** plus corrected the formatting of various screen dumps. They had been reformatted as text at some point.
1.3	24 June 2010	ND	Added a bit more details to the **`-o[ne_at_a_time]`** restore option to explain transactions.
1.4	09 August 2010	ND	Noted that gbak defaults to running a backup or recover according to the first file name parameter supplied. A few minor formatting errors, URLs and some examples were corrected. Also added an example of a meta data only backup and restore.
1.5	31 March 2011	ND	Updated the **`-z`** option to indicate that it *does* carry out a backup.
1.6	11 October 2011	ND	Updated to cover Firebird 2.5 changes. Corrected description of **`-g[arbage_collect]`** switch. Lots of spelling mistakes corrected.
1.7	11 January 2013	ND	Updated to document the use of the stdin and stdout file names in backups and restores, which allow backups to be written to or read from standard input and standard output. A section was added on the use of the above to clone databases without requiring an intermediate dump file. An additional section was also added to show how, using the above in conjunction with SSH, backup and/or restore operations could be carried out on databases where one or both of the databases in question, are remote.
1.8	14 January 2013	ND	Further updates to document the use of the stdin and stdout file names in backups and restores. A section has been added to Gbak Caveats giving more in depth detail about these two special file names.
1.9	11 April 2013	ND	A section has been added to explain how to speedup your backups. A note has been added to the **`-service`** option to explain that it's use is not restriced to remote databases. Syntax errors in some examples corrected.

1.10	1 May 2013	ND	Slight update to the **`-use_[all_space]`** command line switch, to explain how it works in a more understandable manner.
1.11	1 May 2013	ND	A correction to the above change to the **`-use_[all_space]`** command line switch - it affects all subsequent pages as well as the ones created during the restore.

Appendix B: License notice