Firebird Commandline Utilities

Norman Dunbar

Firebird Commandline Utilities

Norman Dunbar

Table of Contents

Chapter 1

Introduction

The Firebird distribution kit installs a number of useful utility programs to assist in the running of your Firebird server & databases. This book introduces you to the various utilities, some of which are more useful than others.

The utilities can all be found in the `/bin` directory (or `\bin` folder on Windows) under the Firebird installation root. On Linux the utilities can be found in `/opt/firebird/bin` while on Windows, the default location is `c:\program files\firebird\firebird_x_y\bin`. Where 'x' and 'y' depend on the version of Firebird installed.

- Fb_lock_print is the utility which prints out details of the internal database lock page.
- Gbak is the database backup & restore utility. It also allows various parameters internal to the database to be changed.
- Gdef is a metadata utility which was removed from Interbase 4.0 and returned in the Open Source version 6. Gdef is probably redundant.
- Gfix allows attempts to fix corrupted databases, starting and stopping of databases, resolving 'in limbo' transactions between multiple database, changing the number of page buffers and so on.
- Gpre is the pre-processor which converts source code, which can be written in a number of languages, containing various embedded SQL 'pseudo code' into correctly formatted calls to the Firebird engine.
- Gsec is the security database manipulation utility. It allows the DBA (or any privileged user) the ability to maintain user accounts for various Firebird databases. Using various options, users can be added, amended or deleted from the security database.
- Gsplit is a filter which allows the limitations on maximum file sizes, found on some operating systems, to be avoided when creating backups of large databases. This utility is supplied on Windows only and, unfortunately, seems not to work. Luckily, gbak allows backup files to be split into multiple parts, so gsplit is not required. On Unix systems there are suitable operating system utilities that can be used instead of gsplit, if required.
- Gstat allows the Firebird administrator the ability to gather statistics about the general health and utilisation of various parts of the database.
- Isql is the interactive utility that allows ad-hoc queries to be run against a Firebird database. It is console based - as are many of the utilities - and is supplied with all distributions of Firebird. Isql is usually the best place to try out your scripts and commands in the first instance.
- Qli is the original Query Language Interpreter which was removed from Interbase 4.0 but returned in Interbase 6.0 because of the decision to Open Source Interbase.
- There are also various shell scripts installed under Linux and other Unix systems.

Note

This book is a work in progress. Each chapter details a separate utility and as each one is completed to my satisfaction, I shall add it to the CVS repository where it will be available for download. In this manner, there will be a slow and gradual build up of hopefully useful manuals.

Chapter 2

Gsec - Firebird Password File Utility

Introduction

Gsec is the security database manipulation utility. It allows the DBA (or any privileged user) the ability to maintain user accounts for various Firebird databases. Using various options, users can be added, amended or deleted from the security database.

Normal users, ie those not endowed with DBA privileges, can only see their own user details from version 2.0 of Firebird. They can, however, change their own passwords with the new version. Previous to the new version, the DBA had to make the changes.

> **Note**
>
> It is possible on some operating systems that users will not be able to run gsec, even if they know the sysdba password. This is because those operating systems allow the system administrator to set filesystem permissions which prevent execution of certain programs and utilities for security reasons.

The Firebird database holds details of all users in a single security database. This is located on the server in a normal Firebird database named `security.fdb` for Firebird 1.5 or `security2.fdb` for Firebird 2.0 onwards. The default locations for this file are :

- `C:\Program Files\Firebird\Firebird_1_5` for Windows.
- `/opt/firebird` for Linux and other Unix systems.

The security database has two tables, users and host_info. The host_info table is empty and the users table holds the details of each user allowed to access any Firebird database. Having said that, database roles and privileges will prevent users logging into and manipulating databases to which they have no rights.

The gsec utility manipulates data in the users table in the security database, and by doing so, allows users to be added, amended and deleted from the system. Not all columns of the users table are able to be displayed, even though they can be amended. The user's password column is never shown by gsec, but you can change it, for example.

Like most of the command line utilities supplied with Firebird, gsec can be run in interactive or batch mode and has a help screen showing all of the utility's options, we'll be seeing that a little later on.

Coming up in this chapter, we have :

- Commandline options for gsec.
- Gsec commands and their parameters.

- Running gsec in interactive or batch modes, both of which allow you to :
  - Display user details.
  - Amend user details.
  - Add new users.
  - Delete existing users.
- Using gsec to administer a remote security database.
- Some caveats, gotchas and foibles of gsec.

Commandline Options

Regardless of the mode that gsec is run in, there are a number of options that can be supplied on the command line. These are :

- **`-user <username>`**

  Allows the username of the sysdba user to be specified if the database is to be modified, or a normal username if the database is to be displayed only. This need not be supplied if `ISC_USER` and `ISC_PASSWORD` environment variables exist and have the correct values.

- **`-password <password>`**

  Supplies the password for the username specified above. This need not be supplied if `ISC_USER` and `ISC_PASSWORD` environment variables exist and have the correct values.

- **`-role <SQL role name>`**

  Allows the specification of the role to be used by the connecting user.

- **`-database <server:security database name>`**

  You can specify the full pathname of a security database to gsec and this will allow you to remotely administer the users for that server. The whole parameter should be enclosed in quotes if there are any spaces in the path to the security database.

- **`-z`**

  Displays the version number of the gsec utility.

- **`-help`**

  Help displays the following screen of information :

```
gsec utility - maintains user password database

command line usage:
  gsec [ <options> ... ] <command> [ <parameter> ... ]

interactive usage:
  gsec [ <options> ... ]
  GSEC>
  <command> [ <parameter> ... ]

available options:
    -user <database administrator name>
    -password <database administrator password>
    -role <database administrator SQL role name>
    -database <security database>
    -z

  available commands:
    adding a new user:
      add <name> [ <parameter> ... ]
    deleting a current user:
      delete <name>
    displaying all users:
      display
    displaying one user:
      display <name>
    modifying a user's parameters:
      modify <name> <parameter> [ <parameter> ... ]
    help:
      ? (interactive only)
      help
    displaying version number:
      z (interactive only)
    quit interactive session:
      quit (interactive only)

  available parameters:
    -pw <password>
    -uid <uid>
    -gid <uid>
    -fname <firstname>
    -mname <middlename>
    -lname <lastname>
```

Gsec Commands

After the assorted options, comes the command that you wish to run. The following commands are available in both batch and interactive modes, but for interactive mode the leading minus sign is not required.

- **`-add <name> [ <parameter> ... ]`**

  This command adds a new user to the database. You may optionally add other details such as first, middle and last names plus a password for the new user, all in the same **`add`** command. Alternatively, you may add a user then **`modify`** it to fill in the missing details.

- **`-delete <name>`**

  This command removes the named user from the database. All details of the user are removed and cannot be undone unless you add the user back again.

- **`-display [ <name> ]`**

  This command displays the details of a single named user, or all users in the database. The password is never displayed.

- **`-modify <name> <parameter> [ <parameter> ... ]`**

The <name> option is how you wish the user to be known when connecting to Firebird databases. Some of the above commands take parameters and these are one, or more, of the following :

- **`-pw <password>`**

  This parameter lets you supply a new password for the user. If you omit the password, the current one will be removed and the user will be unable to login to any Firebird databases at all. The password can be up to 8 characters long, but when supplying one to gsec, or logging into databases, the characters after the eighth are simply ignored.

- **`-uid <uid>`**

- **`-gid <gid>`**

  `-uid` and **`-gid`** are used on some POSIX systems to enter the Unix userid and groupid as found in `/etc/passwd` and `/etc/group` configuration files. If not supplied, these default to zero.

- **`-fname [ <first name> ]`**

  This parameter allows you to store the user's first name in the database. This helps when identifying users from their login name - which may be abbreviated. You can delete a first name by not supplying a name.

- **`-mname [ <middle name> ]`**

  This parameter allows you to store the user's middle name in the database. This helps when identifying users from their login name - which may be abbreviated. You can delete a middle name by not supplying a name.

- **`-lname [ <lastname> ]`**

  This parameter allows you to store the user's last name in the database. This helps when identifying users from their login name - which may be abbreviated. You can delete a last name by not supplying a name.

Interactive Mode

To run gsec in interactive mode, start the utility using the command line :

```
C:\>gsec -user sysdba -password masterkey
GSEC>
```

The GSEC> prompt shows that the utility is waiting for a command. The -user and -password options are those of the user who wishes to manipulate the security database. Obviously, the username supplied must be a valid sysdba user if updates are to be carried out. Normal users may only read the database.

To exit gsec in interactive mode, the quit command is used :

```
GSEC> quit
C:\>
```

The following sections show how to carry out various commands in interactive mode. It is assumed that you are already running the utility as a sysdba user.

Displaying User Details

To display all users in the security database the command, and it's output are :

```
GSEC> display
     user name                      uid   gid     full name
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0      Norman  Dunbar
EPOCMAN                             0     0      Benoit Gilles Mascia
GSEC>
```

To display details of a single user, pass the username as a parameter to the **`display`** command.

```
GSEC> display epocman
     user name                      uid   gid     full name
------------------------------------------------------------------------
EPOCMAN                             0     0      Benoit Gilles Mascia
GSEC>
```

If you enter the name of a non-existent user as a parameter of the **`display`** command, nothing is displayed and gsec remains in interactive mode.

```
GSEC> display alison
GSEC>
```

Adding New Users

When adding a new user in interactive mode, nothing is displayed to confirm that the user was indeed added. You need to use the **`display`** or **`display <name>`** commands to make sure that the user was added successfully.

```
GSEC> add newuser -pw newuser -fname New -lname User
GSEC>
```

```
GSEC> display newuser
     user name                      uid   gid     full name
------------------------------------------------------------------------
NEWUSER                             0     0      New  User
GSEC>
```

Deleting Existing Users

When deleting a user in interactive mode, there is no confirmation that the user has been deleted. You should use the **`display`** or **`display <name>`** command to check.

```
GSEC> delete newuser
GSEC>
```

```
GSEC> display
     user name                      uid   gid     full name
```

```
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0      Norman  Dunbar
EPOCMAN                             0     0      Benoit Gilles Mascia
GSEC>
```

If, on the other hand, you try to delete a non-existing user, gsec will display an error message, and exit.

```
GSEC> delete newuser
record not found for user: NEWUSER

C:\>
```

Amending Existing Users

Existing users can have one or more of their password, first name, middle name or lastname amended. There is no confirmation that your modification has worked, so you must use one of the **`display`** commands to determine how well it worked.

```
GSEC> modify norman -pw newpassword
GSEC>
```

```
GSEC> modify norman -mname MiddleName -fname Fred
GSEC>
```

```
GSEC> display norman
     user name                      uid   gid     full name
------------------------------------------------------------------------
NORMAN                              0     0      Fred MiddleName Dunbar
GSEC>
```

If you wish to remove one or more of a user's attributes, don't pass a (new) value for that attribute.

```
GSEC> modify norman -mname -fname -lname
```

```
GSEC> display norman
     user name                      uid   gid     full name
------------------------------------------------------------------------
NORMAN                              0     0
```

Now I can be known as 'the man with no name', just like Clint Eastwood !

Help

The **`help`** command, in interactive mode, displays the same help screen as shown above.

Version Information

The version of gsec can be obtained using the z command.

```
GSEC> z
```

```
gsec version  WI-V1.5.0.4306 Firebird 1.5
GSEC>
```

Batch Mode

Note

In the following descriptions of batch mode operations, assume that I have set the `ISC_USER` and `ISC_PASSWORD` environment variables. This allows gsec to be run without always having to specify the **`-user`** and **`-password`** switches. This in turn reduces the amount of code on the command line, which means that when this XML file is rendered into pdf, all the commandline will fit on the width of an A4 page.

It is not secure to have these variables set all the time, so don't do it !

Warning

If you are using gsec from Firebird version 1.5 (and possibly version 1.0 as well) then when you are running in batch mode, you may think that you can check the result of an operation by checking `%ERRORLEVEL%` in Windows, or `$?` in various flavours of Unix. This doesn't work. The result is always zero.

In gsec from Firebird version 2.0 onwards, this problem is fixed and the exit code will be zero for everything was ok, or a non-zero value for error conditions.

In batch mode, the command line to run gsec is as follows :

```
     gsec [ <options> ... ] <command> [ <parameter> ... ]
```

Displaying User Details

To display all users in the security database the command, and its output are :

```
C:\>gsec -display
     user name                      uid   gid     full name
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0      Norman  Dunbar
EPOCMAN                             0     0      Benoit Gilles Mascia
```

To display details of a single user, pass the username as a parameter to the **`display`** command.

```
C:\>gsec -display epocman
     user name                      uid   gid     full name
------------------------------------------------------------------------
EPOCMAN                             0     0      Benoit Gilles Mascia
```

Adding New Users

When adding a user in batch mode, there is no confirmation that the user has been added. You should use the **`-display`** or **`-display <name>`** command to check.

```
C:\>gsec -add newuser -pw newuser -fname New -lname User
```

```
C:\>gsec -display
     user name                      uid   gid     full name
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0      Norman  Dunbar
NEWUSER                             0     0      New  User
EPOCMAN                             0     0      Benoit Gilles Mascia
```

Deleting Existing Users

When deleting a user in batch mode, there is no confirmation that the user has been deleted. You should use the **`-display`** or **`-display <name>`** command to check.

```
C:\>gsec -delete newuser
```

```
C:\>gsec -display
     user name                      uid   gid     full name
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0      Norman  Dunbar
EPOCMAN                             0     0      Benoit Gilles Mascia
```

Amending Existing Users

Existing users can have one or more of their password, first name, middle name or lastname amended.

```
C:\>gsec -modify norman -pw newpassword
```

```
C:\>gsec -modify norman -mname MiddleName -fname Fred
```

```
C:\>gsec -display
     user name                      uid   gid     full name
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0      Fred MiddleName Dunbar
EPOCMAN                             0     0      Benoit Gilles Mascia
```

If you wish to remove one or more of a user's attributes, don't pass a (new) value for that attribute.

```
C:\>gsec -modify norman -mname -fname -lname
```

```
C:\>gsec -display
     user name                      uid   gid     full name
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0
EPOCMAN                             0     0      Benoit Gilles Mascia
```

Now nobody knows who I am :o)

Version Information

The version of gsec can be obtained using the **`-z`** command. However, note that it leaves you in interactive mode on completion. It doesn't exit like the other batch mode commands do, so you have to use the interactive **`quit`** command to exit. There is a way around this problem as shown in the following. The first part shows the problem.

```
C:\>gsec -z
gsec version  WI-V1.5.0.4306 Firebird 1.5
GSEC>
```

The solution is to have a small file containing the command **`quit`** and force gsec to read this file when it needs user input, as follows.

```
C:\>copy con fred
quit
^Z
        1 file(s) copied.
```

```
C:\>gsec -z <fred
gsec version  WI-V1.5.0.4306 Firebird 1.5
GSEC>
C:\>
```

This could be a good idea for any of the commands which leave you 'stuck' in the interactive mode when you thought you were running in batch mode. By redirecting input from a command file, gsec will read a line of text from that file any time it requires user input. By forcing it to read the **`quit`** command, you make it exit.

Note

The -z command doesn't need a **`-user`** and **`-password`**, it will display the version details and then tell you that you don't have a username/password - but you can safely ignore this message.

Running Gsec Remotely

Gsec can be used to administer the security database on a remote server. To do this you must supply the remote security database name on the commandline as shown in the following example which connects my Windows XP client version of gsec to my Linux server named Ganymede and allows me to manage the users on my Linux server.

```
C:\>gsec -database ganymede:/opt/firebird/security.fdb
         -user sysdba -password masterkey
GSEC>
```

Note

In the above example, I have split the full commandline over two lines. This is to prevent it 'falling off' the right side of the page when this chapter is rendered as a PDF document. The whole command should, and must, be typed on a single line.

Also note that if there are spaces in the database name, you must enclose the whole parameter in double quotes.

Once connected to the remote security database, you can manipulate users in the normal manner in either interactive or batch modes as described above.

The version of gsec provided in Firebird 2.0 can be used to maintain the security database on previous versions of Firebird and it is hoped, Interbase from version 6.0 upwards. However, under version 2.0 of Firebird, the format of the security database will be changed and because of this, gsec from an older version cannot be used to maintain the security database for Firebird 2.0.

Gsec caveats

The following is a brief list of gotchas and funnies that I have detected in my own use of gsec. 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.

Normal Versus Privileged Users

Only a sysdba user can update the security database. Normal users can run the gsec utility, but can only list the contents under Firebird 1.5. The following shows what happens when trying to update the database when running gsec as a normal user.

```
C:\>gsec -user norman -password norman
GSEC> add myuser -pw mypassword
add record error
no permission for insert/write access to TABLE USERS
```

A normal users can only display details from the security database.

```
C:\>gsec -user norman -password norman -display
     user name                      uid   gid     full name
------------------------------------------------------------------------
SYSDBA                              0     0
NORMAN                              0     0      Norman  Dunbar
EPOCMAN                             0     0      Benoit Gilles Mascia
```

Note

From Firebird version 2 onwards, there are slight changes to the above. Users are now able to change their own passwords and can no longer display details of other users that may be present in the security database.

The above user, running under Firebird 2.0 would see the following :

```
C:\>gsec -user norman -password norman -display
     user name                      uid   gid     full name
------------------------------------------------------------------------
NORMAN                              0     0      Norman  Dunbar
```

Differences Between Batch And Interactive Mode

The gsec commands apply to both modes of operation, however, when running in batch mode, you must prefix the command name with a minus sign (-) or you will get an error message similar to the following :

```
C:\>gsec -user sysdba -password masterkey display
invalid parameter, no switch defined
error in switch specifications
GSEC>
```

Note also that you will be left in interactive mode when an error occurs. The correct commandline should have a minus in front of the **`display`** command, as follows :

```
C:\>gsec -user sysdba -password masterkey -display
     user name                     uid   gid     full name
------------------------------------------------------------------------
SYSDBA                             0     0
NORMAN                             0     0      Norman  Dunbar
EPOCMAN                            0     0      Benoit Gilles Mascia
```

This time, gsec performed its duties, displayed all known users and quit from the utility.

> **Warning**
>
> If environment variables `ISC_USER` and `ISC_PASSWORD` have been defined, and this isn't a very good idea for security reasons, gsec can be run without passing the **`-user`** or **`-password`** options.

> **Warning**
>
> As with all of the command line utilities, it is best to use the version of the gsec utility that was supplied with your database.

Batch Mode Exit Codes

When running gsec under windows, you can trap the exit code in `%ERRORLEVEL%` and check it to determine the success or failure of the last command executed.

When your operating system is Unix - whatever flavour - the exit code is to be found in the `$?` variable.

Unfortunately, using the version of gsec supplied with Firebird 1.5, it appears that gsec always exits with a zero and this makes it quite unsuitable to build into a properly error trapped batch script on either system. Sad but true.

From version 2.0 of Firebird, this has been corrected and an exit code of zero indicates success while non-zero values indicate failures.

Errors In Batch Mode Swap To Interactive Mode

Sometimes, when running in batch mode, an error condition in gsec will result in gsec switching over to interactive mode. This is not very useful if you started gsec in batch mode from a script, because your script will just sit there waiting on something to be typed.

Chapter 3

Gfix - Firebird Database Housekeeping Utility

Introduction

Gfix allows attempts to fix corrupted databases, starting and stopping of databases, resolving 'in limbo' transactions between multiple database, changing the number of page buffers and so on. gfix is a general purpose tool for system administrators (and database owners) to use to make various 'system level' changes to their databases.

Almost all the gfix commands have the same format when typed on the command line:

gfix [commands and parameters] database_name

The commands and their options are described in the following sections. The database name is the name of the *primary* database file which for a single file database is simply the database name and for multi-file databases, it is the first data file added.

Coming up in this chapter, we have:

- Command line options for the gfix database utility.
- Shadow file handling.
- Cache and buffer handling.
- Transaction management.
- Cache management.
- Starting and stopping a database.
- And much, much more ...

Command Line Options

Running gfix without a command (or an invalid command) results in the following screen of helpful information being displayed:

```
invalid switch --help
please retry, specifying an option
plausible options are:
  -activate      activate shadow file for database usage
  -attach        shutdown new database attachments
  -buffers       set page buffers <n>
  -commit        commit transaction <tr / all>
  -cache         shutdown cache manager
  -full          validate record fragments (-v)
  -force          force database shutdown
  -housekeeping   set sweep interval <n>
  -ignore         ignore checksum errors
  -kill           kill all unavailable shadow files
  -list           show limbo transactions
  -mend           prepare corrupt database for backup
  -mode           read_only or read_write
  -no_update      read-only validation (-v)
  -online         database online <single / multi / normal>
  -prompt         prompt for commit/rollback (-l)
  -password       default password
  -rollback       rollback transaction <tr / all>
  -sql_dialect    set database dialect n
  -sweep          force garbage collection
  -shut           shutdown <full / single / multi>
  -two_phase      perform automated two-phase recovery
  -tran           shutdown transaction startup
  -use            use full or reserve space for versions
  -user           default user name
  -validate       validate database structure
  -write          write synchronously or asynchronously
  -z              print software version number

 qualifiers show the major option in parenthesis
```

Gfix Commands

Note

In the following discussion, I use the full parameter names in all examples. This is not necessary as each command can be abbreviated. When the command is shown with '[' and ']' in the name then these are the optional characters.

For example, the command **`-validate`** is shown as **`-v[alidate]`** and so can be specified as **`-v`**, **`-va`**, **`-val`** and so on up to the full **`-validate`** version.

For almost all of the options in the following sections, two of the above command line options will be required. These are **`-u[ser]`** and **`-pa[ssword]`**. These can be supplied for every command as parameters on the command line, or can be configured once in a pair of environment variables.

- **`-u[ser] username`**

  Allows the username of the SYSDBA user, or the owner of the database to be specified This need not be supplied if the `ISC_USER` environment variable has been defined and has the correct value.

- **`-pa[ssword] password`**

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

To define the username and password as environment variables on a Linux system:

```
linux> export ISC_USER=sysdba
linux> export ISC_PASSWORD=masterkey
```

Alternatively, on Windows:

```
C:\> set ISC_USER=sysdba
C:\> set ISC_PASSWORD=masterkey
```

Warning

This is very insecure as it allows anyone who can access your session the ability to perform DBA functions that you might not want to allow.

- **`-u[ser]`** default user name
- **`-pa[ssword]`** default password

If you have not defined the above environment variables, some commands will not work unless you supply **`-u[ser]`** and **`-pa[ssword]`** on the command line. For example:

```
linux> gfix -validate my_employee
linux> Unable to perform operation. You must be either SYSDBA -
or owner of the database
```

Note

The line that starts with 'Unable to perform' above, has had to be split to fit on the page of the PDF file. In reality, it is a single line.

However, passing the username and password works:

```
linux> gfix -validate my_employee -user sysdba -password masterkey
```

Warning

Anyone on the same Linux or Unix server where you are running the above command may be able to use the **ps** command and view the password you have used.

You will notice, hopefully, that some commands do not give any printed output at all. gfix, in the main, only reports when problems are encountered. Always check the response code returned by gfix to be sure that it worked. However, see the caveats section below for details because it looks like the response code is always zero.

Note

When logging into a database on a remote server, you will always be required to pass the **`-u[ser]`** and **`-pa[ssword]`** parameters.

Shadow Files

A shadow file is an additional copy of the primary database file(s). More than one shadow file may exist for any given database and these may be activated and de-activated at will using the gfix utility.

The following descriptions of activating and de-activating shadow files assume that a shadow file already exists for the database. To this end, a shadow was created as follows:

```
linux> isql my_employee;
SQL> create shadow 1 manual '/home/norman/firebird/shadow/my_employee.shd1';
SQL> create shadow 2 manual '/home/norman/firebird/shadow/my_employee.shd2';
SQL> commit;
SQL> show database;
Database: my_employee
 Owner: SYSDBA
 Shadow 1: "/home/norman/firebird/shadow/my_employee.shd1" manual
 Shadow 2: "/home/norman/firebird/shadow/my_employee.shd2" manual
...
SQL> quit;
```

It can be seen that the database now has two separate shadow files created, but as they are manual, they have not been activated. We can see that shadows are in use if we use gstat as follows:

```
linux> gstat -header my_employee | grep -i shadow
Shadow count 2
```

Note

Sometimes, it takes gstat a while to figure out that there are shadow files for the database.

Note

Shadow file details can be found in the RDB$FILES table within the database.

Activating Shadows

The command to activate a database shadow is:

gfix -ac[tivate] <shadow_file_name>

This makes the shadow file the new database file and the users are able to continue processing data as normal and without loss.

In the event that your main database file(s) become corrupted or unreadable, the DBA can activate a shadow file. Once activated, the file is no longer a shadow file and a new one should be created to replace it. Additionally, the shadow file should be renamed (at the operating system prompt) to the name of the old database file that it replaces.

Warning

It should be noted that activating a shadow while the database itself is active can lead to corruption of the shadow. Make sure that the database file is really unavailable before activating a shadow.

Once a shadow file has been activated, you can see the fact that there are active shadows in the output from gstat:

```
linux> gstat -header my_employee | grep -i shadow
Shadow count 2
Attributes   active shadow, multi-user maintenance
```

Note

The DBA can set up the database to automatically create a new shadow file in the event of a current shadow being activated. This allows a continuous supply of shadow files and prevents the database ever running without one.

Killing Shadows

The command to kill *all unavailable* database shadows, for a specific database, is:

gfix -k[ill] database_name

In the event that a database running with shadow files loses a shadow, or a shadow becomes unusable for some reason, the database will stop accepting new connections until such time as the DBA kills the faulty shadow and, ideally, creates a new shadow to replace the broken one.

The following (contrived) example, shows what happens when the database loses a shadow file and an attempt is made to connect to that database. There are two sessions in the following example, one is connected to the database while the second deletes a shadow file and then tries to connect to the database. The command line prompts shows which of the two sessions we are using at the time.

First, the initial session is connected to the database and can see that there are two shadow files attached:

```
linux_1>isql my_employee
Database: my_employee
SQL> show database;
Database: my_employee
   Owner: SYSDBA
Shadow 1: "/home/norman/firebird/shadow/my_employee.shd1" manual
Shadow 2: "/home/norman/firebird/shadow/my_employee.shd2" manual
 ...
```

In the second session, we delete one of the shadow files, and then try to connect to the database

```
linux_2> rm /home/norman/firebird/shadow/my_employee.shd2
linux_2> isql_my_employee
Statement failed, SQLCODE = -901
lock conflict on no wait transaction
-I/O error for file "/home/norman/firebird/shadow/my_employee.shd2"
-Error while trying to open file
-No such file or directory
-a file in manual shadow 2 in unavailable
Use CONNECT or CREATE DATABASE to specify a database
SQL> quit;
```

The second session cannot connect to the database until the problem is fixed. The DBA would use the **gfix -k[ill]** command to remove details of the problem shadow file from the database and once completed, the second (and subsequent) sessions would be able to connect.

```
linux_2> gfix -kill my_employee

linux_2> isql my_employee
Database: my_employee
SQL> show database;
Database: my_employee
```

```
   Owner: SYSDBA
Shadow 1: "/home/norman/firebird/shadow/my_employee.shd1" manual
...
```

The database now has a single shadow file where before it had two. It is noted, however, that gstat still shows the database as having two shadows, even when one has been removed.

```
linux> gstat -header my_employee | grep -i shadow
Shadow count 2
Attributes   active shadow, multi-user maintenance
```

Note

In addition to the above strange result, if I subsequently DROP SHADOW 1 and COMMIT, to remove the remaining shadow file, gstat now shows that the shadow count has gone up to three when it should have gone down to zero!

Set Database Page Buffers

The database cache is an area of RAM allocated to store (cache) database pages in memory to help improve the efficiency of the database performance. It is far quicker to read data from memory that it is to have to physically read the data from disc.

The size of the database cache is dependent on the database page size and the number of buffers allocated, a buffer is the same size as a database page, and whether the installation is using Classic or Superserver versions of Firebird.

In a Classic Server installation, each connection to the database gets its own relatively small cache of 75 pages while Superserver creates a much larger cache of 2,048 pages which is shared between all the connections.

The command to set the number of cache pages is:

gfix -b[uffers] BUFFERS database_name

This command allows you to change the number of buffers (pages) allocated in RAM to create the database cache.

You cannot change the database page size in this manner, only the number of pages reserved in RAM. One parameter is required which must be numeric and between 50 (the minimum) and 131,072 (the maximum).

The setting applies only to the database you specify. No other databases running on the same server are affected.

The following example shows the use of gstat to read the current number of buffers, the gfix utility being used to set the buffers to 4,000 pages and gstat being used to confirm the setting. The value of zero for page buffers indicates the default setting for the server type is in use.

Note

You can use the gstat command line utility to display the database details with the command line: **gstat -header db_name** however, to run gstat, you need to be logged into the server - it cannot be used remotely.

```
linux> gstat -header my_employee | grep -i "page buffers"
Page buffers    0
```

```
linux> gfix -buffers 4000 my_employee

linux> gstat -header my_employee | grep -i "page buffers"
Page buffers 4000
```

Limbo Transaction Management

Limbo transactions can occur when an application is updating two (or more) databases at the same time, in the same transaction. At COMMIT time, Firebird will prepare each database for the COMMIT and then COMMIT each database separately.

In the event of a network outage, for example, it is possible for part of the transaction to have been committed on one database but the data on the other database(s) may not have been committed. Because Firebird cannot tell if these transactions (technically sub-transactions) should be committed or rolled back, they are flagged as being in limbo.

Gfix offers a number of commands to allow the management of these limbo transactions.

> **Note**
>
> The following examples of limbo transactions are based on Firebird 1.5 and have kindly been provided by Paul Vinkenoog. Because of the limitation of my setup, I am unable to create limbo transactions in my current location.
>
> In the spirit of consistency, however, I have renamed Paul's servers and database locations to match the remainder of this document.

Listing Limbo Transactions

The gfix command **`-l[ist]`** will display details of transactions that are in limbo. If there is no output, then there are no transactions in limbo and no further work need be done. The command is:

gfix -l[ist] database_name

An example of listing limbo transactions is shown below. This command is run against the local database on the server named linux where a multi-database transaction had been run connected to databases linux@my_employee and remote:testlimbo. Both of these database names are aliases.

```
linux> gfix -list my_employee
Transaction 67 is in limbo.
 Multidatabase transaction:
 Host Site: linux
 Transaction 67
has been prepared.
 Remote Site: remote
 Database path: /opt/firebird/examples/testlimbo.fdb
```

If the command is run against the remote database then nothing will be listed because that database does not have any limbo transactions - the transaction that went into limbo, when the network failed, for example, was initiated on the local database.

You may also supply the **`-p[rompt]`** option to the command and you will be prompted to COMMIT or ROLLBACK each detected limbo transaction. In this case, the command would be:

gfix -l[ist] -p[rompt] database_name

An example of this is shown below.

```
linux> gfix -list -prompt my_employee
Transaction 67 is in limbo.
 Multidatabase transaction:
 Host Site: linux
 Transaction 67
has been prepared.
 Remote Site: remote
 Database path: /opt/firebird/examples/testlimbo.fdb
Commit, rollback or neither (c, r, or n)?
```

Committing Or Rolling Back

When a limbo transaction has been detected, the DBA has the option of committing or rolling back one or more of the transactions reported as being in limbo.

When more than one transaction is listed, the DBA can either commit or roll back all transactions in limbo, or a specific transaction number.

The following commands show the **`-c[ommit]`** option being used, but the **`-r[ollback]`** option applies as well, it all depends on what the DBA is trying to achieve.

To commit every limbo transaction on the database, the following command would be used:

gfix -commit all database_name

If the DBA wanted to commit a single transaction, then the command would change to the following:

gfix -commit TXN database_name

Where TXN is the transaction number to be committed.

When either of these options are user, there is no feedback from gfix to advise you that the commit actually worked. You would need to rerun the **gfix -list** command to make sure that all, or the selected, limbo transactions had indeed gone.

You cannot commit or rollback a transaction that is not in limbo. If you try , the following will occur:

```
linux> gfix -commit 388 my_employee
failed to reconnect to a transaction in database my_employee
transaction is not in limbo
-transaction 388 is active
unknown ISC error 0
```

When committing or rolling back all limbo transactions, the **`-p[rompt]`** option can be specified. It is, however, not permitted when processing a single transaction. An example of using the **`-p[rompt]`** option has been shown above under listing limbo transactions.

Automatic Two-phase Recovery

Gfix can be used to perform automatic two-phase recovery. The command for this is **`-t[wo_phase]`** and, like **`-c[ommit]`** and **`-r[ollback]`** above, requires either 'all' or a transaction number.

The output of the **`-l[ist]`** command shows what will happen to each listed transaction in the event that the DBA runs the **`-t[wo_phase]`** command.

The command also takes the **`-p[rompt]`** option, as above, when used to process all transaction.

The command line to carry out automatic two-phase recovery is:

gfix -t[wo_phase] TXN database_name or

gfix -t[wo_phase] all database_name

As above, TXN is a single transaction number from the list of limbo transactions.

> **Note**
>
> Paul has noted that when using the **`-c[ommit]`**, **`-r[ollback]`** or **`-t[wo_phase]`** options, the output is exactly the same and appears to show that these three are all just synonyms for the **`-l[ist] -p[rompt]`** pair of options. This occurred whether or not Paul used the transaction number, 67, or 'all' in the command line.

Cache Manager

When the help page for gfix is displayed there is a message in the output for the **`-ca[che]`** option which states:

```
...
-ca[che]      shutdown cache manager
...
```

However, when called this option simply displays the help page again. It is assumed that, at version 2.0 of Firebird, this option does not yet work.

The question that immediately springs to my mind is, if we can shutdown the cache manager with this option, how do we start it back up again?

Changing The Database Mode

Databases can be set to run in one of two modes, read only - where no updates are permitted, and read/write - where both reading and writing of data is permitted. By default, Firebird creates read/write databases and as such, all read/write databases must be placed on a file system which allows writing to take place.

Should you wish to put a Firebird database on a CD, for example, you wouldn't be able to do so. After a new database has been populated with data it can be changed to read only mode, and then used on a CD (or other read only file systems) with no problems.

Note

Firebird uses SQL internally to maintain its internal structures with details about transactions, for example, and this is the reason that a database must be placed on a read/write file system regardless of whether only SELECT statements are run or not.

Note

Only databases in dialect 3 can be changed to read only mode.

The command to set the required mode for a database is:

gfix -mo[de] MODE database_name

The command takes two parameters, the MODE which must be one of the following:

- **`read_only`** - the database cannot be written to.
- **`read_write`** - the database can be written to.

The meaning of the two modes should be quite meaningful.

The second parameter is a database name to apply the mode change to.

The following example shows how to put a database into read only mode, and then change it back again. The example also shows what happens when you try to update the database while running in read only mode.

```
linux> gfix -mode read_only my_employee

linux> isql my_employee
Database: my_employee

SQL> create table test(stuff integer);
Statement failed, SQLCODE = -902
Dynamic SQL Error
-attempted update on read-only database

SQL> quit;

linux> gfix -mode read_write my_employee

linux> isql my_employee
Database: my_employee

SQL> create table test(stuff integer);

SQL> show table test;
STUFF      INTEGER Nullable

SQL> quit;
```

If there are any connections to the database in read/write mode when you attempt to convert the database to read only, the attempt will fail as shown below.

```
linux> gfix -mode read_only my_employee
lock time-out on wait transaction
-lock time-out on wait transaction
```

```
-object my_employee is in use
linux> echo $?
0
```

Warning

As with many failures of gfix, the response code returned to the operating system is zero.

Setting The Database Dialect

The dialect of the database is simply a term that defines the specific features of the SQL language that are available when accessing that database. There are three dialects at present (Firebird version 2.0), these are:

- Dialect 1 stores date and time information in a DATE data type and has a TIMESTAMP data type which is identical to DATE. Double quotes are used to delimit string data. The precision for NUMERIC and DECIMAL data types is less than a dialect 3 database and if the precision is greater than 9, Firebird stores these as DOUBLE PRECISION. INT64 is not permitted as a data type.
- Dialect 2 is available only on the Firebird client connection and cannot be set in the database. It is intended to assist debugging of possible problems with legacy data when migrating a database from dialect 1 to 3. This dialect cannot be set for a database using gfix. (See below.)
- Dialect 3 databases allow numbers (DECIMAL and NUMERIC data types) to be stored as INT64 when the precision is greater than 9. The TIME data type is able to be used and stores time data only. The DATE data type stores on date information. Double quotes can be used but only for identifiers that are case dependent, not for string data which has to use single quotes.

The command to change the SQL dialect for a database is:

gfix -s[ql_dialect] DIALECT database_name

The DIALECT parameter is simply 1 or 3.

The following example changes a database to use dialect 3 which will allow many newer features of SQL 92 to be used.

```
linux> gfix -sql_dialect 3 my_employee

linux> gstat -header my_employee | grep dialect
Database dialect    3

linux> gfix -sql_dialect 1 my_employee

linux> gstat -header my_employee | grep dialect
Database dialect    1
```

Because you cannot use gstat remotely, you may also use the isql command SHOW SQL DIALECT from a remote location to see which dialect your client and database are using, as follows:

```
remote> isql my_employee -user norman -password whatever
Database: my_employee

SQL> show sql dialect;
Client SQL dialect is set to: 3 and database SQL dialect is: 3
```

Although dialect 2 is possible on the client, trying to set a dialect of 2 will fail on the server as the following example shows.

```
linux> gfix -sql_dialect 2 my_employee
Database dialect 2 is not a valid dialect.
-Valid database dialects are 1 and 3.
-Database dialect not changed.
```

To set dialect 2 for your *client* connection, you use isql as follows:

```
linux> isql my_employee
Database: my_employee

SQL> set sql dialect 2;
WARNING: Client SQL dialect has been set to 2 when connecting -
to Database SQL dialect 3 database.

SQL> show sql dialect;
Client SQL dialect is set to: 2 and database SQL dialect is: 3
```

> **Note**
>
> The WANRING line above has had to be split to fit on the page of the PDF version of this manual. In reality, it is a single line of text.

Database Housekeeping And Garbage Collection

Garbage

Garbage, for want of a better name, is the detritus that Firebird leaves around in the database after a rollback has been carried out. This is basically a copy of the row(s) from the table(s) that were being updated (or deleted) by the transaction prior to the rollback.

Because Firebird uses multi-generational architecture, every time a row is updated or deleted, Firebird keeps a copy in the database. These copies use space in the pages and can remain in the database for some time.

In addition to taking up space in the database, these old copies can lead to increased transaction startup times.

There are two types of garbage:

- Remnants from a committed transaction.
- Remnants from an aborted (rolled back) transaction.

These remnants are simply older copies of the rows that were being updated by the respective transactions. The differences are that:

- Whenever a subsequent transaction reaches garbage from a committed transaction, that garbage is automatically cleared out.
- Rolled back garbage is never automatically cleared out.

This means that on a database with a lot of rolled back transactions, there could be a large build up of old copies of the rows that were updated and then rolled back.

Firebird will automatically sweep through the database and remove the remnants of rolled back transactions and this has two effects:

- The database size is reduced as the old copies of rows are deleted.
- The performance of the database may be affected while the sweep is in progress.

> **Note**
>
> One other method of clearing out old rolled back transactions' garbage is simply to carry out a database backup.

In the Super Server version of Firebird 2.0, garbage collection has been vastly improved. There are now three different ways of operation and these are configurable by setting the *GCPOLICY* parameter in the `firebird.conf` configuration file. By default, Super Server uses *combined* while Classic Server uses *cooperative*. The other option is *background*.

> **Note**
>
> Classic Server ignores the setting and always uses cooperative garbage collection.

Cooperative Garbage Collection

This is the default setting, indeed the only setting, that Classic Server uses. In this mode, the normal operation - as described above - takes place. When a full scan is performed (perhaps during a backup) old versions of the rows are deleted at that point in time.

Background garbage Collection

Super Server has, even since before version 1.0, performed background garbage collection where the server informs the garbage collector about old versions of updated and deleted rows when they are ready to be cleaned up. This helps avoid the need to force a full scan of each record in the database tables to get the garbage collector to remove these old versions.

When all rows in a table are read by the server, any old record versions are flagged to the garbage collector as being ready to be cleared out. They are not deleted by the scanning process as in the cooperative method. The garbage collector runs as a separate background thread and it will, at some point, remove these older record versions from the database.

Combined Garbage Collection

This is the default garbage collection method for Super Server installations. In this mode, both the above methods are used together.

Setting Sweep Interval

The default sweep interval for a new database is 20,000. The sweep interval is the *difference* between the *oldest interesting transaction* or OIT and the next transaction number.

Note

This doesn't mean that every 20,000 transaction a sweep will take place. It will take place when the *difference* between the OIT and the next transaction is greater than the sweep interval.

An interesting transaction is one which has not yet committed. It may be still active, in limbo or may have been rolled back.

The sweep facility runs through the database and gets rid of old rows in tables that are out of date. This prevents the database from growing too big and helps reduce the time it takes to start a new transaction on the database.

Note

If you find that starting a new transaction takes a long time, it may be a good idea to run a manual sweep of the database in case the need for a sweep is causing the hold-up.

You can check if a manual sweep may be required by running the gstat utility to check the database header page and extract the oldest and next transaction numbers from the output. If the gap is small (less than the sweep interval) then a manual sweep may be in order. Alternatively, the SHOW DATABASE command in isql will also show the details you need.

A manual sweep can be run by using the **`-s[weep]`** command. (See below).

To alter the database's automatic sweep interval, use the following command:

gfix -h[ousekeeping] INTERVAL database_name

The INTERVAL parameter is the new value for the sweep interval. The database name parameter is the database upon which you wish to alter the setting for automatic sweeping. The following example shows the setting being changed from the default to a new value of 1,000.

```
linux> gfix -h 1000 my_employee

linux> gstat -header my_employee | grep Sweep
Sweep interval:     1000
```

Manual Garbage Collection

If automatic sweeping has been turned off, or only runs rarely because of the sweep interval setting, the DBA can manually force a sweep to be performed. The command to carry out this task is:

gfix -s[weep] [-i[gnore]] database_name

This command will force the garbage left over from old rolled back transactions to be removed, reducing the database size and improving the performance of new transactions.

The **`-i[gnore]`** option may be supplied. This forces Firebird to ignore checksum errors on database pages. This is not a good idea and should rarely need to be used, however, if your database has suffered some problems it might be necessary to use it.

The following example shows a manual database sweep being implemented:

```
linux> gfix -sweep my_employee
```

Disabling Automatic Sweeping

If you set the sweep interval to zero then automatic sweeping will be disabled. This implies that there will be no automatic housekeeping done so your database performance will no suffer as a result of the processing requirements of the automatic sweep.

If you disable sweeping you are advised to run a manual sweep at regular intervals when the database is quiet. Alternatively, simply make sure that you take regular backups of the database and as this is something you should be doing anyway, it shouldn't be a problem.

Database Startup and Shutdown

Note

The first part of this section describes the shutdown and startup options up to Firebird 2.0. There is a separate section at the end which discusses the new *states* for starting and stopping a database using Firebird 2.0 onwards.

Database Shutdown

If there is maintenance work required on a database, you may wish to close down that database under certain circumstances. This is different from stopping the Firebird server as the server may well be running other databases which you do not wish to affect.

The command to close a database is:

gfix -shut OPTION TIMEOUT database_name

The TIMEOUT parameter is the time, in seconds, that the shutdown must complete in. If the command cannot complete in the specified time, the shutdown is aborted. There are various reasons why the shutdown may not complete in the given time and these vary with the mode of the shutdown and are described below.

The OPTION parameter is one of the following:

- **`-at[tach]`** - prevents new connections.
- **`-tr[an]`** - prevents new transactions.
- **`-f[orce]`** - simply aborts all connections and transactions.

When a database is closed, the SYSDBA or the database owner can still connect to perform maintenance operations or even query and update the database tables.

Note

If you specify a long time for the shutdown command to complete in, you can abort the shutdown by using the **`-online`** command (see below) if the timeout period has not completed.

Preventing New Connections

`-at[tach]` : this parameter prevents any new connections to the database from being made with the exception of the SYSDBA and the database owner. The shutdown will fail if there are any sessions connected after the timeout period has expired. It makes no difference if those connected sessions belong to the SYSDBA, the database owner or any other user. Any connections remaining will terminate the shutdown with the following details:

```
linux> gfix -shut -attach 5 my_employee
lock conflick on no wait transaction
-database shutdown unsuccessful
```

Anyone other than the SYSDBA or database owner, attempting to connect to the database will see the following:

```
linux> isql my_employee -user norman -password whatever
Statement failed, SQLCODE = -901
database my_employee shutdown
Use CONNECT or CREATE DATABASE to specify a database
SQL>
```

Connections in the database will still be able to start new transactions or complete old ones.

Preventing New Transactions

`-tr[an]` : prevents any new transactions from being started and also prevents new connections to the database. If there are any active transactions after the timeout period has expired, then the shutdown will fail as follows:

```
linux> gfix -shut -tran 5 my_employee
lock conflick on no wait transaction
-database shutdown unsuccessful
```

If any user connected to the database being shutdown with the **`-tr[an]`** tries to start a new transaction during the shutdown timeout period, the following will result:

```
SQL> select * from test;
Statement failed, SQLCODE = -902
database /home/norman/firebird/my_employee.fdb shutdown in progress
Statement failed, SQLCODE = -902
database /home/norman/firebird/my_employee.fdb shutdown in progress
Statement failed, SQLCODE = -901
Dynamic SQL Error
-SQL error code = -901
-invalid transaction handle (expecting explicit transaction start)
```

Force Closure

`-f[orce]` : shuts down with no regard for the connection or transaction status of the database. No new connections or transactions are permitted and any active sessions are terminated along with any active transactions.

Anyone other than SYSDBA or the database owner trying to connect to the database during the timeout period will not be able to connect successfully or start any (new) transactions.

Be nice to your users, use the **`-f[orce]`** option with great care.

> **Warning**
>
> There is a bug in Classic Server which still exists at version 2.0. The bug is such that the **`-f[orce]`** option behaves in exactly the same way as the **`-at[tach]`** option.

Starting a Database

Once all maintenance work required on a database has been carried out, you need to restart the database to allow normal use again. (See shutdown option above for details of closing a database.)

The **`-o[nline]`** command allows a database to be restarted. It takes a single parameter which is the database name as follows:

gfix -o[nline] database_name

The following example shows a closed database being started.

```
linux> gfix -online my_employee
```

New Startup and Shutdown States in Firebird 2.0

The above discussion of stopping and starting a database apply to all versions of the server up to version 2.0. From 2.0 the commands will work as described above, but a new *state* has been added to define exactly how the database is to be stopped or started. The commands change from those described above to the following:

gfix -shut STATE OPTION TIMEOUT database_name

gfix -o[nline] STATE database_name

STATE is new in Firebird 2.0 and is one of the following:

- **`normal`** - This is the default state for starting the database backup. It allows connections from any authorised users - not just SYSDBA or the database owner. This option is not accepted for shutdown operations.
- **`multi`** - this is the default mode as described above. When the database is shutdown as above, or using the multi state, then *unlimited* connections can be made by the SYSDBA or the database owner. No other connections are allowed.
- **`single`** - Similar to the multi option above, but only *one* SYSDBA or database owner connection is allowed.
- **`full`** - Shutdown and don't allow *any* connections from anyone, even SYSDBA or the database owner. This is not an acceptable option for starting up a database.

> **Note**
>
> There is no leading dash for the state parameters, unlike the command itself and the **`-shut`** OPTION.

There is a hierarchy of states for a database. The above list shows them in order with normal at the top and full at the bottom.

This hierarchy is important, you cannot *shutdown* a database to a *higher or equal* level that it currently is, nor can you *startup* a database to a *lower or equal* level.

If you need to identify which level a database is currently running at, gstat will supply the answers. The following example puts a database fully online then progressively shuts it down to fully offline. At each stage, gstat is run to extract the Attributes of the database.

```
linux> gfix -online normal my_employee
linux> gstat -header my_employee | grep Attributes

        Attributes

linux> gfix -shut multi -attach 0 my_employee
linux> gstat -header my_employee | grep Attributes

        Attributes              multi-user maintenance

linux> gfix -shut single -attach 0 my_employee
linux> gstat -header my_employee | grep Attributes

        Attributes              single-user maintenance

linux> gfix -shut full -attach 0 my_employee
linux> gstat -header my_employee | grep Attributes

        Attributes              full shutdown

linux>
```

Database Page Space Utilisation

When a database page is being written to, Firebird reserves 20% of the page for future use. This could be used to extend VARCHAR columns that started off small and then were updated to a longer value, for example.

If you wish to use all the available space in each database page, you may use the -use command to configure the database to do so. If you subsequently wish to return to the default behaviour, the -use command can be used to revert back to leaving 20% free space per page.

Note

Once a page has been filled to 'capacity' (80 or 100%) changing the page usage setting will not change those pages, only subsequently written pages will be affected.

The **`-use`** command takes two parameters as follows:

gfix -use USAGE database_name

The USAGE is one of:

- **`full`** : use 100% of the space in each database page.
- **`reserve`** : revert to the default behaviour and only use 80% of each page.

The following example configures a database to use all available space in each database page:

```
linux> gfix -use full my_employee
linux> gstat -header my_employee | grep Attributes
Attributes no reserve
```

The following example sets the page usage back to the default:

```
linux> gfix -use reserve my_employee
linux> gstat -header my_employee | grep Attributes
Attributes
```

If you are using full page utilisation then the Attributes show up with 'no reserve' in the text. This doesn't appear for normal 80% utilisation mode.

Database Validation and Recovery

Database Validation

Sometimes, databases get corrupted. Under certain circumstances, you are advised to validate the database to check for corruption. The times you would check are:

- When an application receives a *database corrupt* error message.
- When a backup fails to complete without errors.
- If an application aborts rather than shutting down cleanly.
- On demand - when the SYSDBA decides to check the database.

Note

Database validation requires that you have exclusive access to the database. To prevent other users from accessing the database while you validate it, use the **gfix -shut** command to shutdown the database.

When a database is validated the following checks are made *and corrected* by default:

- Orphan pages are returned to free space. This updates the database.
- Pages that have been misallocated are reported.
- Corrupt data structures are reported.

There are options to perform further, more intensive, validation and these are discussed below.

Default Validation

The command to carry out default database validation is:

gfix -v[alidate] database_name

This command validates the database and makes updates to it when any orphan pages are found. An orphan page is one which was allocated for use by a transaction that subsequently failed, for example, when the application aborted. In this case, committed data is safe but uncommitted data will have been rolled back. The page appears to have been allocated for use, but is unused.

This option updates the database and fixes any corrupted structures.

Full Validation

By default, validation works at page level. If no need to go deeper and validate at the record level as well, the command to do this is:

gfix -v[alidate] -full database_name

using this option will validate, report and update at both page and record level. Any corrupted structures etc will be fixed.

Read-only Validation

As explained above, a validation of a database will actually validate and update the database structures to, hopefully, return the database to a working state. However, you may not want this to happen and in this case, you would perform a read only validation which simply reports any problem areas and does not make any changes to the database.

To carry out a read only validation, simply supply the **`-n[o_update]`** option to whichever command line you are using for the validation. To perform a full validation, at record and page level, but in reporting mode only, use the following command:

gfix -v[alidate] -full -n[o_update] database_name

On the other hand, to stay at page level validation only, the command would be:

gfix -v[alidate] -n[o_update] database_name

Ignore Checksum Errors

Checksums are used to ensure that data in a page is valid. If the checksum no longer matches up, then it is possible that a database corruption has occurred. You can run a validation against a database, but ignore the checksums using the **`-i[gnore]`** option.

This option can be combined with the **`-n[o_update]`** option described above and applies to both full and default validations. So, to perform a full validation and ignore checksums on a database, but reporting errors only, use the following command:

gfix -v[alidate] -full -i[gnore] -n[o_update] database_name

Alternatively, to carry out a page level validation, ignoring checksum errors but updating the database structures to repair it, the command would be:

gfix -v[alidate] -i[gnore] database_name

Ignoring checksums would allow a corrupted database to be validated (unless you specify the **`-n[o_update]`** option) but it is unlikely that the recovered data would be usable, if at all, present.

Database Recovery

If the database validation described above produces no output then the database *structures* can be assumed to be valid. However, in the event that errors are reported, you may have to repair the database before it can be used again.

Recover a Corrupt Database

The option required to fix a corrupted database is the gfix -m[end] command. However, it cannot fix all problems and may result in a loss of data. It all depends on the level of corruption detected. The command is:

gfix -m[end] database_name

This causes the corruptions in data records to be ignored. While this sounds like a good thing, it is not. Subsequent database actions (such as taking a backup) will not include the corrupted records, leading to data loss.

Important

The best way to avoid data loss is to make sure that you have enough regular backups of your database and to regularly carry out test restorations. There is no point taking backups every night, for example, if they cannot be used when required. Test always and frequently.

Database Write Mode

Many operating systems employ a disc cache mechanism. This uses an area of memory (which may be part of your server's overall RAM or may be built into the disc hardware) to buffer writes to the hardware. This improves the performance of applications that are write intensive but means that the user is never certain when their data has actually been written to the physical disc.

With a database application, it is highly desirable to have the data secured as soon as possible. Using Firebird, it is possible to specify whether the data should be physically written to disc on a COMMIT or simply left to the operating system to write the data *when it gets around to it*.

To give the DBA or database owner full control of when data is written, the **gfix -w[rite]** command can be used. The command takes two parameters:

gfix -write MODE database_name

The MODE parameter specifies whether data would be written immediately or later, and is one of:

- **sync** - data is written synchronously. This means that data is flushed to disc on COMMIT. This is safest for your data.
- **async** - data is written asynchronously. The operating system controls when the data is actually written to disc.

If your system is highly robust, and protected by a reliable UPS (uninterruptable Power Supply) then it is possible to run asynchronously but for most systems, synchronous running is safest this will help prevent corruption in the event of a power outage or other uncontrolled shutdown of the server and/or database.

Note

Firebird defaults to synchronous mode (forced writes enabled) on Linux, Windows NT, XP, 2000, 2003 and Vista.

This command has no effect on Windows 95, 98 and ME.

Warning

Cache flushing on Windows servers (up to but not including Vista - which has not been confirmed yet) is unreliable. If you set the database to **`async`** mode (forced writes disabled) then it is possible that the cache will never be flushed and data could be lost if the server is never shutdown tidily.

Warning

If your database was originally created with Interbase 6 or an early beta version of Firebird then the database will be running in asynchronous mode - which is not ideal.

Version Number

The **`-z`** option to gfix simply prints out the version of the Firebird utility software that you are running. It takes no parameters as the following example (running on Linux) shows.

```
linux> gfix -z
gfix version LI-V2.0.0.12748 Firebird 2.0
```

Caveats

This section summarises the various problems that you may encounter from time to time when using gfix. They have already been discussed above, or mentioned in passing, but are explained in more details here.

Shadows

The gstat seems to take some time to respond to the addition of shadow files to a database. After adding two shadows to a test database, gstat still showed that there was a Shadow count of zero.

Even worse, after killing the second shadow file and running the DROP SHADOW command in isql to remove the one remaining shadow file, gstat decided that there were now three shadow files in use.

Response Codes Are Always Zero

Even under version 2 of gfix, it appears that any command which fails to complete without an error returns a response of 0 to the operating system.

For example, the following shows two attempts to shut down the same database, the second one should fail - it displays an error message - but still returns a zero response to the operating system. This makes it impossible to built correctly error trapped database shutdown scripts as you can never tell whether it actually worked or not.

```
linux> gfix -shut -force 5 my_employee
linux> echo $?
0

linux> gfix -shut -force 5 my_employee
Target shutdown mode is invalid for database -
"/home/norman/firebird/my_employee.fdb"
linux> echo $?
0
```

Note

In the above, the line warning about the shutdown mode being incorrect has had to be split over two lines. They are, in fact, a single line of text. The PDF generation requires long lines be split in this manner.

Force Closing a Database

Under classic server, using the **`-f[orce]`** option to the **`-shut`** command acts exactly the same as the **`-at[tach]`** option.

Limbo Transactions

There are a couple of problems with limbo transactions as discovered by Paul in his testing.

Limbo Transaction Options - All The Same?

When processing limbo transactions, it appears under Firebird 1.5 at least, that the **`-l[ist] -p[rompt]`** option is called regardless of whether you use **`-c[ommit]`**, **`-r[ollback]`** or **`-t[wo_phase]`**. The outcome is the same regardless of whether the DBA specifies a specific transaction number or 'all' on the command line - a prompt is given with the option to commit, rollback or neither.

Limbo Transactions - Can Be Backed Up

Paul's testing of limbo transactions revealed that it is possible to make a backup of a database with limbo transactions. This backup can then be used to create a new database and the limbo transactions will still be able to be listed. This applies to a filesystem copy of the database and to version 1.5 of Firebird.

If you attempt to list the limbo transactions in the copy database *and* the original database has been deleted, renamed or has been set to read-only, then gfix will present you with a request to supply the correct path to the original database

```
linux>cd /home/norman/firebird
linux>cp my_employee.fdb my_new_employee.fdb
linux> mv my_employee.fdb my_old_employee.fdb

linux> gfix -list /home/norman/firebird/my_new_employee.fdb
Transaction 67 is in limbo.
Could not reattach to database for transaction 67.
Original path: /home/norman/firebird/my_employee.fdb

Enter a valid path: /home/norman/firebird/my_old_employee.fdb

 Multidatabase transaction:
 Host Site: linux
 Transaction 67
has been prepared.
 Remote Site: remote
 Database path: /opt/firebird/examples/testlimbo.fdb
```

In the above example, the original database my_employee.fdb was first of all copied using the operating system command **cp** to my_new_employee.fdb and then renamed to my_old_employee.fdb.

Gfix was then run on the copy named my_new_employee.fdb and it noted the limbo transaction. However, it could not find the original database file as it had been renamed, so gfix prompted for the path to the original database file. When this was entered, gfix happily listed the details.

Warning

This implies that if you have a database with limbo transactions and you copy it using the operating system utilities and subsequently run gfix against the new database, it is possible to have gfix fix limbo transactions in the original database file and not in the one you think it is updating - the copy.

It is also a good warning about making copies of databases without using the correct tools for the job.

Chapter 4

Gsplit - Firebird Backup File Splitting Filter

Introduction

In the past, many operating systems imposed a limit which defined how large a single file could become. This limit was 2 Gb on some systems, and 4 Gb on others. For example, on HP-UX 10.20 or 11.00 Unix systems, the maximum file size is 2 Gb unless the file system has the *largefiles* option turned on. This limit still exists on some operating systems.

Gsplit is a filter utility introduced with Interbase 5.0 which allows the output file from the gbak utility (when backing up a database) to be split into a number of chunks, or a number of chunks to be joined together and used to restore a database. Up until Interbase 5.0, dump files were limited to 2 Gb by the gbak utility itself - even if running on a system which allowed files to be 4 Gb.

Coming up in this chapter, we have :

- Command line options for gsplit.
- Gsplit options and their parameters.
- Splitting backups using gsplit.
- Joining backup chunks using gsplit.

Note

From Interbase 6.0, gsplit is no longer required as gbak allows large files to be split directly. The details for gsplit given here are for reference only and you are advised to use gbak to split large backup files even if gsplit is supplied with your Firebird release.

Gsplit is only supplied with the Windows version of Firebird 1.5, it is not supplied with the Linux version. Linux doesn't require a separate utility as it can split files using the **split** command as well as using gbak.

Warning

In testing with Firebird 1.5, on Windows XP Home, gsplit doesn't seem to work and always returns error 9.

Because of the problems in getting gsplit to work correctly, as you shall see in the remainder of this chapter, you are advised to use the splitting and joining facilities of the gbak utility itself rather than trying to make gsplit work for you.

Gsplit Command line Options

Gsplit has three command line options, although, strictly speaking, the **-help** option isn't really valid. These are :

- **-split_bk_file <parameters>**

  Specifies that gsplit is to be used to split the output from gbak into a number of different files as part of a database backup. This option can be shortened as required, provided that at least **-s** is specified.

- **-join_bk_file <parameters>**

  Specifies that gsplit is to be used to rejoin a number of files and use the result as input to gbak as part of a database restore. This option can be shortened as required, provided that at least **-j** is specified.

- **-help**

  Using the -help option, specifying an illegal option, or omitting all options, displays the following information :

```
gsplit: invalid option '-help'
gsplit: Command Line Options Are:
gsplit -S[PLIT_BK_FILE] <file> <size>{k|m|g} [... <file> [<size>{k|m|g}]]
  or gsplit -J[OINT_BK_FILE] <file> [... <file>]
gsplit: option can be abbreviated to the unparenthesized characters
gsplit: Exiting before completion due to errors
```

Note

Take note of the error in the above help text. The correct command line option to join multiple sections of a large backup file is not **-JOINT_BK_FILE** as shown, but is in fact **-JOIN_BK_FILE**.

Gsplit Command Parameters

Both main gsplit command line options require parameters. When splitting a backup, the parameters are :

- **-S[PLIT_BK_FILE] <file> <size>{k|m|g} [... <file> [<size>{k|m|g}]]**

  The first parameter is the first filename, followed by the maximum size it is allowed to be. You may specify the size in kilobytes, megabytes or gigabytes. There should not be any spaces between the digit(s) and the size specifier letter. There must be a space between the filename and the size however.

  The remainder of the parameters specify the second and subsequent filenames and sizes, however, the final file need not have a size specified as it will be used to take up the remaining bytes after the other files have been filled to capacity. If there is a size specified, it is simply ignored but no error or warning messages are displayed.

  If you have a backup file which will be 4 Gb in size and you only ask for two files, each 1 Gb in size, gsplit will ignore the size of the final file and just keep filling it up until the backup is complete.

The utility prevents files of less than 1 Mb and will produce an error if you attempt to specify a file smaller than this.

Note

Gsplit correctly specifies a Kilobyte as 1024 bytes, a Megabyte as 1024 Kilobytes and a Gigabyte as 1024 Megabytes.

- **`-J[OIN_BK_FILE] <file> [... <file>]`**

  To join files together and use them to restore a database, you simply specify the filenames in the correct order. If they are not in the correct order, gsplit will complain and the restore will be abandoned.

Splitting Backups

To run gsplit , you need to use it as a filter on the command line for gbak, as the following example shows :

```
C:\>gbak -b norman.fdb stdout |
    gsplit -split norman_1.fbk 1m norman_2.fbk 1m norman_3.fbk
```

Warning

The command above assumes that `ISC_USER` and `ISC_PASSWORD` have been defined. For the sake of this demonstration, this is acceptable, but in a real system, consider the security implications before defining these variables.

In addition, the above command line has been split over two lines to allow the pdf generation of this manual to work. In reality, the command must be typed on a single line.

It is unfortunate that the utility doesn't seem to work, as the following shows :

```
C:\>gbak -b norman.fdb stdout |
    gsplit -split norman_1.fbk 1m norman_2.fbk 1m norman_3.fbk
fail to read input from ib_stdin, errno = 9
gsplit: progam fails to generate multi-volumn back-up files
Done with volume #0, "stdout"
        Press return to reopen that file, or type a new
        name followed by return to open a different file.
  Name:^C
```

If you type a filename at the prompt it will simply be used as a destination for a full dump of the database, so be careful not to overwrite anything important. I tend to hit **CTRL-C** at this point to avoid problems.

The utility has actually created the first file in the above list, `norman_1.fbk`, and written 100 bytes to a special header which identifies it as a gsplit created file.

Warning

The command above assumes that `ISC_USER` and `ISC_PASSWORD` have been defined. For the sake of this demonstration, this is acceptable, but in a real system, consider the security implications before defining these variables.

In addition, the above command line has been split over two lines to allow the pdf generation of this manual to work. In reality, the command must be typed on a single line.

Note

The spelling errors in 'program' and 'volume' above are as produced by the utility.

Joining Backup Files

Had the above backup actually worked, the command to restore a backup from a number of files created by gsplit would be as follows :

```
C:\>gsplit -join norman_1.fbk norman_2.fbk norman_3.fbk |
    gbak -c stdin create_norman.fdb
```

Warning

The above command line has been split over two lines to allow the pdf generation of this manual to work. In reality, the command must be typed on a single line.

If you have a number of split backup files created using gbak itself and not filtered through gsplit, you cannot use gsplit to stitch them together for a restore as the following example shows :

```
C:\>gsplit -join norman_1.fbk norman_2.fbk norman_3.fbk |
    gbak -c stdin create_norman.fdb
gsplit: expected GSPLIT description record
gsplit: Exiting before completion due to errors
gsplit: progam fails to join multi-volumn back-up files
gbak: ERROR: expected backup description record
gbak: Exiting before completion due to errors
```

Warning

The above command line has been split over two lines to allow the pdf generation of this manual to work. In reality, the command must be typed on a single line.

It appears that gsplit and gbak have different header information in the backup files and the two are not compatible.

Note

The spelling errors in 'program' and 'volume' above are as produced by the utility.

Chapter 5

Assorted Firebird Scripts On Linux/Unix Systems

Introduction

After a successful installation of Firebird, the `/opt/firebird/bin` directory will contain a number of useful shell scripts. This chapter gives details of what these scripts are for and how they are used.

Coming up in this chapter, we have :

- changeDBAPassword.sh
- createAliasDB.sh
- fb_config
- changeRunUser.sh
- restoreRootRunUser.sh
- changeGdsLibraryCompatibleLink.sh

> **Note**
>
> The above list is correct at the time of writing and applies to Firebird 1.5 Super Server installed on a Linux system. Other flavours of Unix may have different scripts. I am currently unable to discuss those potential changes as I do not have access to other Unix systems.

Changing SYSDBA's Password

The `changeDBAPassword.sh` script allows the password for the SYSDBA user to be changed and various startup scripts etc to have their ownership changed accordingly. The script is run in non-interactive mode as part of the installation process to create an initial randomly generated password which is stored in the `/opt/firebird/SYSDBA.password` file. The password is subsequently used in the startup script `/etc/rc.d/init.d/firebird`, which also has a symlink set up to point to `/etc/init.d/firebird`. The security database `/opt/firebird/security.fdb` is also updated with the new password.

> **Caution**
>
> The script must be run as the root user, and, when run, changeDBAPassword.sh will prompt you for the current SYSDBA password and then for a new password. Both of these will appear on the display so for enhanced security, don't allow anyone to look over your shoulder when you run the script.
>
> After the script has been run, the file `/opt/firebird/SYSDBA.password` will contain the password in plain text, so make sure that this file is not readable by anyone except root.

The following is an example of running the script to change the SYSDBA password from 'masterkey' to 'biroguin' which being a made up word, should be less crackable or guessable.

```
# cd /opt/firebird/bin
# ./changeDBAPassword.sh
Please enter current password for SYSDBA user : masterkey
Please enter new password for SYSDBA user : biroguin
GSEC> GSEC>

Running ed to modify /etc/init.d/firebird
#
```

Caution

Whenever you change the SYSDBA password using the gsec utility, you should also change the startup script file as well. To ensure a complete update, always use this script when changing the SYSDBA user's password.

Creating Database Alias Names

The `createAliasDB.sh` script allows the creation of a new database to be carried out, and an alias for that database to be created in the file `/opt/firebird/aliases.conf`.

Note

The createAliasDB.sh script must be run as the root user.

If your system is not set up properly, the database creation step may fail but the alias will still be added to the aliases file. This could leave you subsequently unable to add the alias properly, as the script checks to ensure that you do not overwrite an existing alias. You will have to manually edit the alias file to remove the broken alias.

To set up new databases, they must be owned by the firebird user, and also owned by the firebird group. The following shows a new directory being created by the root user to allow Firebird databases to be created.

```
# cd /u01
# mkdir databases
# chown firebird:firebird databases
#
```

At this point the directory `/u01/databases` is available for use as a repository for one or more Firebird databases. Obviously, in the above example, the `/u01` directory already existed.

createAliasDB.sh Parameters

To run the `createAliasDP.sh` script, use a commandline similar to the following :

```
# createAliasDB.sh <new_alias> <database_name>
```

The script takes two parameters on the commandline, both of which are mandatory :

- **`new_alias`**

  The first parameter is the new alias you wish to create. This alias must not already exist in the alias file, or an error will be displayed and no further action taken.

- **`database_filename`**

  The second parameter specifies the *full path* to the database file. You must not specify a relative path as this could lead to incorrect database filenames being used at connection time. The script will reject any attempt to pass a relative pathname instead of a full pathname.

  A brand new empty database will be created provided that the name passed to the script doesn't already exist. If the database already exists, only the alias will be created and added to the alias file.

A Helping Hand With Makefiles

The `fb_config` script is intended to be used from within a makefile to supply various settings and compiler options which are accurate and specific to the installation being used.

> **Note**
>
> This script can be run by any user who has been given execute privileges to it. You do not need to be root to use this script.

fb_config Options

To run the `fb_config` script, use a commandline similar to the following :

```
fb_config <option> [ <option> [...]]
```

The script takes one or more options on the commandline :

- **`--help`**

  This parameter displays the list of allowed options. It should not be supplied in addition to other options.

- **`--cflags`**

  This option returns the list of directories where Firebird include files are stored. This is required by the C and C++ compilers to allow compiler #include directives to be correctly resolved. On my own system, this option returns '`-I/opt/firebird/include`'.

- **`--libs`**

  This option returns the list of directories where Firebird libraries are located and a list of those libraries that are required to be linked by the linker to create a client server Firebird application. This option returns '`-L/opt/firebird/lib -lfbclient`' on my system.

- **`--embedlibs`**

This option returns the list of directories where Firebird libraries are located and a list of those libraries that are required to be linked by the linker to create an embedded Firebird application. This option returns '`-L/opt/firebird/lib -lfbembed`' on my system.

- **`--bindir`**

  On my system, this option returns '`/opt/firebird/bin`' as the full path to the Firebird `/bin` directory.

- **`--version`**

  This option returns a three part version string made up of the concatenation of the Firebird build version, a dash, the package version, a dot and the system architecture. My own laptop Linux system returns '`1.5.0.4290-0.i686`'.

The following is a brief excerpt from a makefile which shows how to define two macros, FBFLAGS and FBLIBS, and initialise them to the correct values using fb_config. Note the use of the back tick character (`) rather than a single quote character (').

```
...
FBFLAGS = `fb_config --cflags`
FBLIBS = `fb_config --libs`
...
```

Changing The Firebird Server Run User

There are two versions of the `changeRunUser.sh` script, the one prefixed 'SS' is for Super Server installations and the one prefixed 'CS' is for Classic Server installations. The following describes the Super Server version only.

> **Note**
>
> This script should be run as root.

The `SSchangeRunUser.sh` script allows the user and group, under which the Super Server runs, to be changed. By default, this is now the firebird user and group, however, in previous versions the Firebird server ran as the root user which is undesirable from a system security point of view and allowed databases to be created all over the file system. With the new firebird user, restrictions can be placed on where databases can be created.

The script changes the owing user and group of a number of files in the Firebird installation directory, the logfile and also the startup script `/etc/rc.d.init.d/firebird` which is used to start and stop the Firebird server.

SSchangeRunUser.sh Parameters

To run the script, use a commandline similar to the following :

`SSchangeRunUser.sh <username> <groupname>`

The script takes two parameters on the commandline, both of which are optional as you will be prompted if both are omitted. If you only supply one parameter, it is assumed to be the username and you will be prompted for the groupname.

- **`username`**

  This parameter sets the username under which the Super Server is to run. The supplied value is validated against entries in `/etc/passwd`.

- **`groupname`**

  This parameter sets the groupname under which the Super Server is to run. The supplied value is validated against entries in `/etc/group`.

The following example shows the use of `SSchangeRunUser.sh` to change the owning user and group to firebird. The firebird user and group is actually the default when Firebird is installed so there is no need for you to run the script unless you have changed these details already.

```
# cd /opt/firebird/bin
# ./SSchangeRunUser.sh firebird firebird
Updating /opt/firebird
Updating startup script
Completed
#
```

Restoring root As The Firebird Server Run User

There are two versions of the `restoreRootRunUser.sh` script. The one prefixed 'SS' is for Super Server installations and the one prefixed 'CS' is for Classic Server installations. The following describes the Super Server version only.

> **Note**
>
> This script must be run as root.

This script simply restores the old style installation format whereby the Firebird Super Server runs as the root user and group. This script is simply a wrapper around the `SSchangeRunUser.sh` script, passing root as the username and group name.

Running Embedded Or Client Server Applications

This script, `changeGdsLibraryCompatibleLink.sh`, is available with Classic Server installations only, and is used to change the symlink `libgds.so` to point to the appropriate library for the installation. There are two possible libraries that the symlink can point to :

- `/opt/firebird/lib/libfbclient.so` for client server applications
- `/opt/firebird/lib/libfbembed.so` for embedded server applications.

After installation, the `libgds.so` symlink points to the client server library by default so if you are running an embedded application, you need to run this script to point `libgds.so` at the embedded library instead.

Note

This script must be run as root.

The following example shows how this script is used to change from embedded servfer to client server use :

```
# cd /opt/firebird/bin
# ./changeGdsCompatibleLibraryLink.sh
For classic server there are two optional backward compatible client
libraries. These are libfbclient.so and libfbembed.so.

libfbclient.so) enables your client to be multithreaded but must
                connect to a database via a server.
libfbembed.so)  allows the client to directly open the database file,
                but does not support multithreaded access

Your current setting is:
/usr/lib/libgds.so -> /opt/firebird/lib/libfbembed.so

Which option would you like to choose (client|embed|remove)
                                                       [client] client
#
```

The default option is **`client`** which will recreate the symlink to the client server library, **`embed`** will recreate the symlink to the embedded server, while **`remove`** will remove the symlink altogether.

There are no messages displayed to inform you of the success of the script, however, if you run it again, you will notice the current setting should be different to that displayed when you previously ran the script.

Chapter 6

Still to come ...

As this is a work in progress, please excuse the 'sudden' ending to this book. As I research and document the remaining commandline utilities, I shall add new chapters to this book. Until such time as the book is complete, this chapter will give brief details of work I still have to complete.

- Fb_lock_print is the utility which prints out details of the internal database lock page.
- Gbak is the database backup & restore utility. It also allows various parameters internal to the database to be changed.
- Gdef is a metadata utility which was removed from Interbase 4.0 and returned in the Open Source version 6. Gdef is probably redundant.
- Gfix allows attempts to fix corrupted databases, starting and stopping of databases, resolving 'in limbo' transactions between multiple database, changing the number of page buffers and so on.
- Gpre is the pre-processor which converts source code, which can be written in a number of languages, containing various embedded SQL 'pseudo code' into correctly formatted calls to the Firebird engine.
- Gstat allows the Firebird administrator the ability to gather statistics about the general health and utilisation of various parts of the database.
- Isql is the interactive utility that allows ad-hoc queries to be run against a Firebird database. It is console based - as are many of the utilities - and is supplied with all distributions of Firebird. Isql is usually the best place to try out your scripts and commands in the first instance.
- Qli is the original Query Language Interpreter which was removed from Interbase 4.0 but returned in Interbase 6.0 because of the decision to Open Source Interbase.

Norman Dunbar.

Appendix A: License notice