Using Firebird

(work in progress)

IBPhoenix Editors
Maintenance, additions: Firebird Project

16 July 2007 – Document version 2.0.2

Using Firebird

(work in progress)

16 July 2007 – Document version 2.0.2
by IBPhoenix Editors
Maintenance, additions: Firebird Project

Table of Contents

List of Figures

List of Tables

Chapter 1

About this book

Work in progress!

This document is a work in progress. The Firebird documenters – all volunteers who work on this project in their spare time – will publish more chapters as they go along. As we are but a few, the pace is pretty slow. However, the few chapters that *are* present are up to date and will hopefully be a useful source of information for you.

If you find any errors or inconsistencies, please report them to the maintainers. Email addresses are at the end of the book.

Origins

Using Firebird is not a new book. The IBPhoenix editors wrote it years ago for distribution on their Developer's CD, when Firebird was still at version 1.0. Since then we have seen the arrival of Firebird 1.5 and 2.0, and most of the book is now in serious need of updating. In 2005 the IBPhoenix company decided to open-source the entire 26-chapter manual and hand it over to the Firebird Project for upgrading and maintenance.

We would like to thank IBPhoenix here for making this work freely available.

More documentation

If you're new to Firebird or want to brush up on the basics, read the *Firebird Quick Start Guide* first. There's one available for every version of Firebird. Pick up yours from http://www.firebirdtest.com/en/documentation/. This is also a good starting place to find links to more Firebird documentation.

Chapter 2

About Firebird

Firebird is a powerful, compact client/server SQL relational database management system which can run on a variety of server and client operating systems. Its officially supported platforms are Windows and Linux, but Firebird is also known to run on several other OS's, such as FreeBSD and Apple Macintosh OS/X.

Firebird features a higher level of SQL standards compliance than most other industrial-strength client/server database management systems on the market today, while implementing some powerful language features in the vendor-specific sphere of procedure programming.

Firebird's origins

The product which today we call Firebird has been around, under a variety of names, for well over 20 years. An overview of its interesting and at times stormy history can be found at http://www.firebirdtest.com/en/historical-reference/.

Developed as an ongoing open source project, Firebird is a descendant of Borland's InterBase 6.0 Open Edition code which was released for open source development in July 2000 under the InterBase Public License (IPL).

The Firebird source code tree is maintained on the international open source code foundry, Sourceforge, by a large team of professional developers who donate time and expertise voluntarily to fix, develop and enhance this popular and feature-rich database management software.

The Firebird software products are distributed completely free of registration or deployment fees.

The Firebird Foundation

The Firebird Foundation supports the development of Firebird in several ways, among other things by issuing grants to developers. Many people and companies who find Firebird useful have already become members or sponsors. If you like Firebird, please consider doing the same. Making a one-time donation is also possible. You can find more information at http://www.firebirdtest.com/en/firebird-foundation/.

Overview of Features

Firebird is true client/server software, architected for use in local and wide-area networks. Accordingly, its core consists of two main software programs:

1. The database server, which runs on a network host computer.

2. The client library, through which users on remote workstations connect to and communicate with databases managed by the server.

TCP/IP is the network protocol of choice for Firebird on all platforms, although Windows Networking (NetBEUI) is supported too for networks having Firebird running on a Windows NT, 2000/2003 or XP host server.

It is possible to run both server and client on the same physical machine and have the client connect to the server through TCP/IP local loopback. On Windows machines, a single local client can also connect to a database by sharing inter-process communications memory with the Firebird server. On Linux, even direct I/O to the database file is possible, but only with the so-called *Classic Server* – more on this later.

Firebird Server

Firebird server runs on a number of platforms, including:

- Windows NT 4.0, 2000, and 2003 (Server or Workstation editions)
- Windows 95/98 and ME
- Windows XP (Home, Professional and .NET editions)
- Linux, FreeBSD and several other UNIX-like operating systems
- MacOS X (Darwin)

The *Firebird Embedded Server* is a special variant which contains both the client and the server functionality. You can ship it with your application, unpack it, and it's ready to roll. You'll learn more about its up- and downsides later on in this guide.

Firebird clients

A remote workstation or a local client requires only the shared client library – a dynamic link library on Windows and a shared object on other platforms – and an application program which can pass and receive parameters to and from the library's interface.

Generally, you would also install a copy of the client library on the host server, for use with several of the Firebird command-line utilities and/or any server-based management programs you might use. Many of these utilities can also be run remotely, however. A remote system administrator can manage some of the essential services provided by these utilities by accessing them through a host service controller process.

For Java connectivity, Firebird provides the JDBC/JCA-compliant Jaybird driver. Client applications written against Jaybird can run on any Java-enabled platform, even those that don't support Firebird server. The legacy InterClient Java driver is no longer maintained, due to its severe limitations.

Summary of features

Table 2.1. Summary of features

Firebird Feature	Description
SQL compliance	Firebird conforms to entry-level SQL-92 requirements. It has support for formal, cascading referential integrity constraints, updatable views, and full, left and right outer joins. Client applications can link to the Firebird API, a messenger function library for client-server communication.

Firebird Feature	Description
	The Firebird server supports development of dynamic SQL client applications. It also ships with a host-language precompiler and in-engine language support for embedded SQL development in host languages such as C/C++ and COBOL. Several extended SQL features are also implemented. Some of them (e.g. stored procedures and triggers, SQL roles, and segmented blob support) anticipate SQL99 extensions.
Multiuser database access	Firebird is designed to provide for many clients accessing a single database at the same time. In their turn, client applications can have active connections to several databases simultaneously. Firebird will automatically protect cross-database transactions through a two-phase commit mechanism. Triggers and stored procedures can post *event messages* to inform interested clients of specific events in the database.
User-defined functions	User-defined functions (UDFs) can be written and stored on the server machine in external shared object libraries. Once a UDF is declared to a Firebird database as an external function, it is available to any client application accessing the database, as if it were a native function of the SQL language. This flexibility accounts for the very small footprint of the server engine: Firebird database application solutions are deployed without the extra cargo of a server that supports hundreds of unused functions natively in its engine.
Transactions	Firebird client applications have full control over the starting, committing, and rolling back of transactions. Every transaction exists in its own consistent context, determining isolation from other transactions and resolution of multi-user conflicts at commit time. A transaction's uncommitted view of the state of the database is kept consistent with its initial view and any changes which are made within its own context. Client applications can isolate multiple tasks in separate transactions simultaneously. A single transaction can bridge a task involving an unlimited number of connected databases, with an automatic two-phase commit mechanism to protect integrity, should a database become unavailable before the transaction completes.
Multigenerational architecture	Firebird uses a multi-generational architecture, by which multiple versions of each data row can be created and stored as necessary if a transaction modifies the row. In a background thread, extinct versions are garbage-collected and the current and pending versions are managed, in order to give each transaction a persistent view and to resolve priorities when update conflicts occur. The multi-generational architecture of Firebird means that readers never block writers. Firebird allows any row to be visible to any transaction, even if other transactions have updates pending for it. Readers may of course see another (older) *version* of the row than the writer. The Firebird engine maintains version statistics which it uses, in conjunction with the isolation and lock response attributes of each transaction, to determine which transaction gets priority when conflicting updates are requested.

Firebird Feature	Description
Optimistic row-level locking	In Firebird, user-initiated locking is unnecessary. The engine locks a row to other transactions only when a transaction signals that it is ready to update it. This is known as optimistic row-level locking. This style of locking has great advantages in increasing throughput and reducing serialisation for client tasks, when compared with systems that lock rows, or even entire tables, from the moment the transaction begins.
BLOB filters	Firebird provides the capability for the developer to supply filter code for converting stored BLOBs from one format to another. For example, a BLOB filter could be written to output a text BLOB, stored in RichText format, as XML or HTML; or to output a stored JPEG image in PNG format. The filters, written in the developer's language of choice and compiled for the server platform OS, are stored on the server machine in a shared object library and must be declared to databases that want to use them, exactly like UDF libraries.
Database administration	Firebird comes with various command-line utilities for managing databases and servers. Thanks to its open source character, Firebird is also abundantly supported by open source, freeware and commercial GUI database administration utilities. Using his or her preferred constellation of tools, the database administrator can • manage server security; • make and restore database backups; • perform maintenance tasks; • produce database and lock manager statistics.
Security	Firebird maintains a security database storing user names and encrypted passwords. It is located in the root directory of the server installation and controls access to the server itself and all databases in its physical domain. The SYSDBA account has full, destructive privileges to all databases on the server. Firebird provides the capability to define ROLEs at database level. Within a database, only SYSDBA and the database owner have full privileges; otherwise, all privileges must be granted explicitly to individual users and/or roles. It is possible – and recommended – to define a set of permissions for a role and then grant that role to specific users as required. SYSDBA can add and delete users' accounts names and modify the details of an account, including the password. Passwords, once stored, are not human-readable, even by SYSDBA. Physical database paths can be shielded from the client using *aliases*. Access to database files, external tables, and UDFs can be restricted to explicitly specified filesystem trees only – or even tighter – by setting the appropriate parameters in the configuration file `firebird.conf`. The Firebird server process can – and if possible, should – run as a user other than the system or superuser account (`root`, `Administrator` or `localsystem`). This will limit the damage in the unfortunate event that the server should be hacked.
Backups and restores	Firebird comes with two command-line backup/restore tools, each with its own specific advantages and limitations.

Firebird Feature	Description
	The *gbak* utility backs up a database by dismantling it into a compact structure in which metadata, data and database-level configuration settings are stored separately. It also performs some important housekeeping tasks on the database during the backup process. The generated backup is not readable as a database file; you need gbak again to restore it. In restore mode, gbak can create a new file or overwrite an existing database. Because of the useful tasks it performs, experienced Firebird programmers often use a gbak backup-restore cycle to • erase obsolete record versions; • change the database page size; • convert the database from single- to multifile; • safely transfer a database to another operating system; • upgrade InterBase or Firebird databases to a newer version; • make a metadata-only backup in order to create a new, empty database with the same structure. Several user-friendly GUI front-ends are available for gbak, both as stand-alone tools and as utilities within some of the database administration programs. It is also very simple to set up OS-level scripts, batch files or daemons to perform backups. A more recent tool by the name of *nbackup* lacks most of gbak's housekeeping and compaction features, but has the following advantages: • Incremental backups, which save time and disk space; • Backups at hardware speed; • Backups possible with your own preferred (non-Firebird) tool. Neither backup tool requires exclusive access to the database. Other clients can remain connected and perform operations on the database while the backup is in progress.
Other tools	Firebird ships with several other command-line administration tools, including: *isql* An SQL query utility which can run dynamic SQL (DSQL) and several specialised statements interactively or in batch from a script. This is the tool to use for quick access to information about your metadata and for running data definition scripts. *gfix* A database housekeeping and repair kit for minor corruptions. This tool is often used in combination with some of the utilities in the gbak program for identifying and recovering damaged data. *gsec* A command-line interface to the security database. *gstat*

Firebird Feature	Description
	A utility for printing out the current configuration and statistics of a running database. *fb_lock_print* A utility for printing out the Lock Manager report on a running database.
Services API	Firebird provides a *Services API* which developers can use to perform a number of security and management tasks programmatically (and if needed, remotely). Strictly speaking, the Services API (part of the client library) is the interface to the *Services Manager* (part of the server), but the terms are often used interchangeably.

Classic and Superserver architectures

Firebird server comes in two distinct architectures for managing multiple client connections: *Superserver* and *Classic Server*. For Windows, both architectures are included in a single download. For Linux, there are separate download packages which have either CS or SS in their name, indicating the type of server.

The Classic server starts a separate process for each connection to a database under its control. Each client is allocated its own database cache buffers. Superserver serves many clients simultaneously within a single process. Instead of separate server processes for each connection it uses threads of a single process and pools the database cache buffers for use by all connections.

If you are upgrading from a previous version of Firebird or faced with the choice between Classic and Superserver, the information listed in the comparison table below will help to explain what the differences are and how they affect database operations.

The server architecture does not affect the structure of databases or the way client applications work. Firebird databases built on a Classic server can be operated on by an equivalent Superserver server, and vice versa. The same client library can connect to either server.

In other words, if you begin by installing the Superserver distribution of Firebird on your Linux host machine and, later, decide to change to Classic, any applications you wrote for your Superserver-hosted databases will work unmodified and the databases themselves will continue to behave as they did before.

Comparison of characteristics

The table below gives a quick overview of the main differences between Classic and Superserver. These differences will be discussed in more detail in the subsections that follow.

Table 2.2. Comparison of Classic and Superserver architectures

FEATURE	ARCHITECTURE	
	Classic	**Superserver**
Availability	Linux: All Firebird versions Windows: Firebird 1.5 and higher	All Firebird versions
Executable	`fb_inet_server(.exe)`	`fbserver(.exe)`
Processes	Multiple, on demand, one instance per client connection.	Single server process, each client request is handled in its own thread.
Lock management	`gds_lock_mgr` process.	Implemented as a thread.
Local access on Linux	Fast, direct I/O to the database file is possible. But you can also connect network-style via `localhost`.	Network-style access only.
Local access on Windows	Versions 1.x: network-style access only.	Versions 1.x: single (!) local connections can be made using IPC (IPServer). Network-style local connections are also supported.
	Firebird 2 and higher: both architectures support safe, multiple local connections on Windows machines through XNET.	
Resource use	One cache per process.	One cache space for all clients.
Multiprocessor support	Yes.	No. Performance may drop if not properly configured.
Services Manager + API	Partial in Firebird 1.5, full in 1.5.1 and up.	Full.
Guardian on Windows	On Firebird 2 Classic/Win only, a bug prevents you from using the Guardian if you run Firebird as an *application*.	The Guardian functions with all Windows Superservers, whether run as a service or as an application.
Guardian on Linux	You can't use the Guardian with *any* Firebird Classic version on Linux. This is by design.	The Guardian functions with all Linux Superservers.

Executable and processes

Classic
: Runs on demand as multiple processes. When a client attempts to connect to a Firebird database, one instance of the `fb_inet_server` executable is initiated and remains dedicated to that client connection for the duration of the connection.

Superserver
: Runs as a single process, an invocation of the `fbserver` executable. `fbserver` is started once by the owning user or by a boot script. This process runs always, waiting for connection requests. Even when no

client is connected to a database on the server, `fbserver` continues to run quietly. On Linux, the Superserver process does not depend on `inetd`; it waits for connection requests to the `gds_db` service itself.

Lock management

Classic
For every client connection a separate server process is started to execute the database engine, and each server process has a dedicated database cache. The server processes contend for access to the database, so a Lock Manager subsystem is required to arbitrate and synchronise concurrent page access among the processes.

Superserver
The lock manager is implemented as a thread in the `fbserver` executable. It uses inter-thread communication mechanisms to resolve access issues. Therefore, an external process isn't needed.

Resource use

Classic
Each instance of `fb_inet_server` keeps a cache of database pages in its memory space. While the resource use per client is greater than in Superserver, Classic uses fewer overall resources when the number of concurrent connections is low.

Superserver
Employs one single cache space which is shared by client attachments, allowing more efficient use and management of cache memory when the number of simultaneous connections grows larger.

Local access on Linux

Classic
On Linux only, the Classic architecture permits application processes that are running on the same machine as the database and server to perform I/O on database files directly. Note that this is only possible if the client process has sufficient filesystem-level access rights to the database as well as some other files. Network-style access to the local server (via `localhost` or equivalents) is supported on all systems.

Superserver
You can only connect to local databases via TCP/IP loopback, using `localhost` or any other host name / IP number that points back to the local machine. (Many clients may let you get away with omitting the hostname though, and supply `localhost` to the server automatically.)

Local access on Windows

Classic
In Windows Classic versions prior to Firebird 2, you can only connect to local databases via network loopback, using `localhost` or an equivalent. Firebird 2 and higher support local access through the reliable XNET protocol, which permits multiple simultaneous connections in a safe way.

Superserver
Firebird 1.5 and earlier Superservers use the IPC (IPServer) protocol for single local connections on Windows. This method is not as fast and certainly not as robust as the direct I/O on Linux Classic. Furthermore, IPC needs an internal window to exchange messages. As a consequence, local access on these versions is only available if:

- the Firebird process runs as `Localsystem` (the default), *and*
- the configuration parameter *`CreateInternalWindow`* has not been set to 0 (you would set this to 0 if you want to run multiple servers simultaneously).

Firebird 2 uses a different local protocol – XNET – which doesn't suffer from these restrictions, and supports multiple connections.

Of course if local protocol is disabled you can still connect to any local database via `localhost`, provided TCP/IP is available on your system.

Multiprocessor support

Classic
: Supports SMP (symmetrical multi-processor) systems. This improves the performance in case of multiple unrelated connections.

Superserver
: No SMP support. In fact, Superserver performance may drop significantly on multiprocessor Windows systems as a result of processor swapping. To prevent this from happening, set the *`CpuAffinityMask`* parameter in the configuration file `firebird.conf`.

Services Manager and Services API

Classic
: Versions up to and including 1.5 have a partially implemented Services Manager, supporting tasks like backup/restore, database shutdown etc. over the network. Other service tasks have to be performed locally using the client tools that come with Firebird. Versions 1.5.1 and up have a full Services Manager, just like Superserver.

Superserver
: The Services Manager, present in all Firebird Superserver versions, allows you to perform management tasks (backup/restore, database shutdown, user management, stats, etc.) programmatically. You can connect to the Services Manager over the network and thus perform these tasks remotely.

Use of the Firebird Guardian

The Firebird Guardian is a utility which monitors the server process and attempts to restart the server if it terminates abnormally.

Classic
: Due to a bug in the Guardian, it can't be used with Firebird 2 Classic on Windows if run as an application. If Firebird runs as a service, the Guardian works correctly. Since the Windows 9x–ME line doesn't support services, you can't use the Guardian with Firebird 2 Classic on those systems. This bug does not exist in Firebird 1.5 versions.

  (The Guardian can't be used *at all* with Firebird Classic on Linux, but that's by design, not by accident.)

Superserver
: The Guardian works fine with Superserver on both Linux and Windows, whether as a service or as an application.

Which is better?

In abstract terms, neither architecture is a clear winner. One architecture generally outshines the other under specific workload conditions:

- A single application running on the same machine as the server is faster with the Classic architecture.
- For a Linux application embedded in an appliance, Classic is better, because it provides a single process from application to disk.
- On a single-processor machine, an application with larger numbers of frequently contending clients is faster with Superserver, because of the shared cache.
- On SMP machines, small numbers of clients whose data updates do not impact others' tasks work better in the Classic architecture.

Embedded server

Besides Superserver and Classic, there's Firebird Embedded Server for Windows, which you can download as a separate package. This is not really a different architecture, but a Firebird client plus Superserver rolled into one DLL for ease of deployment. Although it has a number of downsides, it may be an attractive option if you want to include Firebird with your Windows application. More on Embedded Server in the client-server chapter.

System Requirements

Firebird makes efficient use of system resources. Both server and clients are modest in their disk space and memory requirements. Some specific details are provided below.

Server Memory (all platforms)

Table 2.3. Memory Requirements

Firebird server process	When there are no connections, the Firebird server uses around 2–4 Mb of memory, depending on the version.
Client connections	Each client connection uses from 115 Kb to several Mb of additional memory on the server host. The exact load depends on the Firebird version, the structure of the database and the client characteristics.
Database cache	Memory is also needed for database page caching. The default cache size is configurable, in database pages. Superserver shares a single cache among all connections and increases cache automatically when required. Classic creates an individual cache per connection.
Other server tasks	The server uses additional memory for lock management, in-memory sorting, and so on. For some tasks the amount can be configured.

Disk space

Disk space requirements vary somewhat according to platform, architecture and Firebird version.

Table 2.4. Approximate Disk Space Requirements

	Firebird 1.5.x	**Firebird 2**
Complete server installation	9–12 Mb	12–14 Mb
Client library	350 Kb – 2 Mb *	380 Kb – 2.5 Mb *
Command-line tools	1.5 Mb	1.7–2.7 Mb
Temporary server space	Additional disk space is required for temporary storage during operation, e.g. for sorting. Location(s) and maximum amount of space used can be configured according to performance demands and the likely volume and type of data to be handled.	

*The “high end” of the client library range is occupied by Linux Classic clients, which contain a complete Firebird engine.

In addition, third-party database management utilities will require 1 Mb to several dozen Mb, depending on which one(s) you choose.

Minimum machine specifications

> **Note**
>
> Wherever Intel processors are mentioned, the equivalent or better AMD processors can also be used.

Table 2.5. Minimum machine specifications

OS	Version	CPU	RAM
Microsoft Windows	NT 4.0 with Service Pack 6a Windows 95/98/ME Windows 2000 (SP1) / 2003 Windows XP	486DX2 66 MHz (Pentium 100 recommended)	16Mb for client 64Mb for multi-client server
Linux	*1.0:* Red Hat 6.2, TurboLinux 6.0, SuSE 7.0, Mandrake 7.2 *1.5:* glibc 2.2.5, libstdc++ 5.0 RedHat 8.0, Mandrake 9.0, SuSE 8.0 On SuSE 7.3, *first* install `libgcc-3.2-44.i586.rpm` and `libstdc++-3.2-44.i586.rpm`	1.0: Intel 486 1.5: Pentium	16Mb for client 64Mb for multi-client server
Solaris	2.6 or 2.7	SPARC, UltraSPARC	16Mb for client 64Mb for multi-client server
Solaris	?	Intel	32 Mb 64 Mb for multi-client server
Apple Macintosh	Mac OS/X (Darwin)	See distribution notes	See distribution notes
FreeBSD	v.4.x	See distribution notes	See distribution notes
HP-UX	10.0	See distribution notes	See distribution notes

Chapter 3

About Clients and Servers

In this chapter we take a look at the essential pieces of client/server systems as they are implemented in Firebird and examine how applications interact with the client and server.

What is a Firebird client?

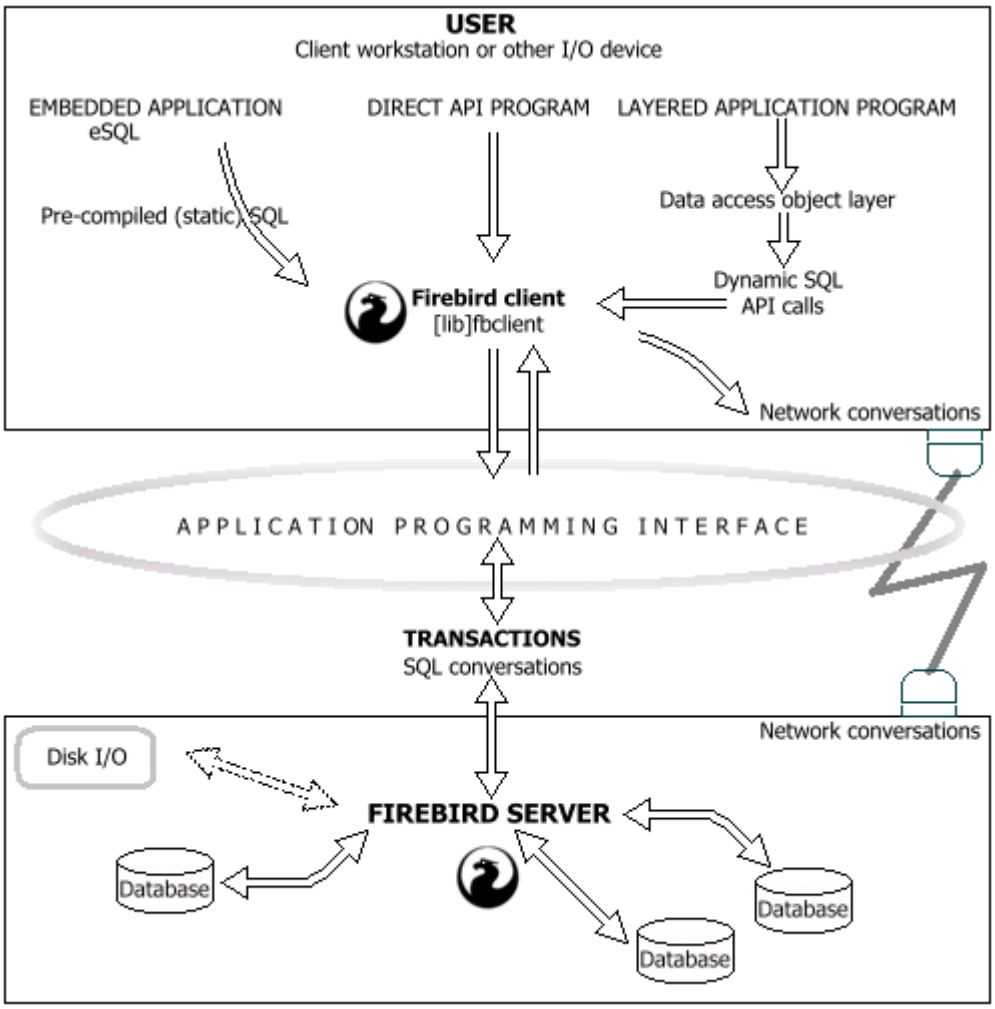
A Firebird client is a program, usually written in a high-level language such as C, C++, Delphi, Java, PHP or Perl, that provides end-user access to the features and tools of the Firebird database management system and to data stored in databases. The isql interactive SQL utility is an example of a client application.

In the client/server model, applications never touch the database physically. Any application process converses with the server through the Firebird client library which resides on the client workstation. It surfaces a programming interface of function call structures known as the Firebird API. This client library must be installed on every user's workstation. Generally, other layers are also involved in the interface between the application program and the Firebird client, providing generic or application-language-specific mechanisms for populating and calling the API functions.

Firebird clients typically reside on remote workstations and connect to a Firebird server running on a host node in a network. Firebird also supports local connection, that is, a client application, the Firebird client library and the Firebird server all executing on the same physical box.

Firebird clients need not run on the same type of hardware and/or operating system as the server they connect to. It is quite common to have a number of Windows 98 or XP workstations talking to a server that runs under Windows NT, 2000 or 2003, or any of several flavours of UNIX or Linux.

Figure 3.1. The Firebird client/server model



The Firebird client library

The Firebird client library provides an Application Programming Interface (API) with functions for connecting to servers and working with databases. The library functions communicate with the server(s) using a dedicated Firebird client/server protocol that sits on top of the general network protocol provided by the OS.

All client applications and middleware must use the API in some way to access Firebird databases. The Firebird API is backwardly compatible with the InterBase API. The *InterBase API Guide* (available at http://www.ibphoenix.com/downloads/60ApiGuide.zip) contains extensive documentation on the use the API in applications. Additional features available in the Firebird API are documented in the Firebird release notes.

Client filenames

The Firebird client library files are `fbclient.dll` (Windows), `libfbclient.so` (Linux network client) and `libfbembed.so` (Linux local client with embedded engine, Classic only). In order not to break certain

existing applications, a copy of `fbclient.dll` with the old name `gds32.dll` can be installed on Windows machines. On Linux, legacy `libgds*` symlinks are installed automatically.

The server

The Firebird server is a program that runs on a machine with which client workstations can communicate by way of a network. Clients connect to databases physically located on this server host machine. The same machine that hosts the executing Firebird server process must host the Firebird databases in its own storage space. Only the server process has direct, filesystem-level access to the database files. The server is fully network-enabled, serving multiple connections simultaneously, in response to requests from other nodes in the network. If the network runs under TCP/IP protocol, the scope of the network is virtually limitless.

In the Superserver architecture, the server process is multi-threaded. In Classic, a new process is started for each connection.

Server tasks

The server's job is to

- regulate access by transactions to individual sets of data;
- ensure that each transaction gets and keeps a consistent view of the permanently stored data which it has requested through the client;
- receive requests to modify or delete a row and either:
  - grant a transaction exclusive write access to the row, or
  - deny access if another transaction already has a write pending;
- maintain the statistics for each database;
- maintain and refer to the metadata for each database, in order to manage transactions, data and “house-cleaning”.

Clients' requests result in the server performing tasks such as

- creating new databases;
- creating new data structures inside databases;
- validating and compiling source code for procedures;
- searching tables for data matching provided criteria;
- collating, sorting and tabulating sets of data;
- passing sets of data back to the requesting client;
- modifying the values of data;
- inserting new data into tables;
- removing (deleting) data;
- executing compiled procedures;
- routing messages to clients.

Multiple servers

Starting at version 1.5, you can have multiple Firebird servers running on the same machine. Firebird 1.5 servers can also coexist with a Firebird 1.0 or InterBase server. Some extra configuration steps are required though.

One thing to be aware of is that Firebird 1.5.*n* under Windows needs the *CreateInternalWindow* configuration parameter to be set to 0 in order to run multiple servers. As a side effect, the local connection protocol is disabled (you can still connect to local databases via `localhost` though). This limitation no longer exists in Firebird 2, and the *CreateInternalWindow* parameter has been removed in that version.

Server filenames

Server binaries on Windows are called `fbserver.exe` (Superserver) and `fb_inet_server.exe` (Classic). On Linux, the same names are used, but without the `.exe` extension.

Client and server combined: Firebird Embedded Server

Embedded server on Windows

The Windows *Embedded Server* is a Superserver engine plus client rolled into one library, called `fbembed.dll`. It is available as a separate download package. The embedded server (introduced with Firebird 1.5) was specifically designed to facilitate deployment of applications that ship with a Firebird server included. Installation is just a matter of unpacking the DLL and a few other files to the desired location. The security database is not used at all: anyone can connect to any database, as long as the user running the application has filesystem-level access rights to the database(s) in question.

You can have multiple embedded servers running at the same time, and you can have multiple apps connecting to the same embedded server. Having a regular server already running isn't a problem either. However, an embedded server locks a database file for its own exclusive use after successful connection. This means that you cannot access the same database from multiple embedded server processes simultaneously (or from any other servers, once an embedded server has locked the file).

The embedded server has no facility to accept any network connections. Only true local access is possible, with a connect string that doesn't contain a host name (not even `localhost`).

Needless to say, this is not for your regular client-server database usage, as it bypasses a lot of security features. It's using Firebird as a desktop database system.

> **Note**
>
> A Firebird embedded server DLL can also be used as a network client. That is, if a regular Firebird server is listening for connections on the target computer, you can connect to databases on that system using a network-style connect string like `inca:C:\School\Databases\Pupils.fdb`.
>
> This also implies that if a regular server is active on your local computer, you can connect to local databases through that regular server with your embedded server as a client using the "`localhost:`" syntax. This may seem contradictory to what has been said before about the absence of network support, but bear in mind that if you connect to `localhost` (or any other host), you are not using the embedded *server*; you're only using the *client* part of `fbembed.dll` to connect to another server. You'll have to provide a valid user name and password for this to work.

Embedded server deployment

First, download the Embedded server kit from SourceForge. It's typically named `Firebird-`*`n.n.n.xxxx`*`_embed_win32.zip`, with *`n.n.n.xxxx`* the Firebird version and build number.

After unzipping, you'll find the embedded server `fbembed.dll` in the root directory of the package, along with some other files. Additionally, there are three subdirectories `doc`, `intl` and `udf`.

To make your application work with the embedded server:

1. Copy `fbembed.dll` into your application directory. Rename it to `fbclient.dll` or `gds32.dll`, depending on what your application expects as a Firebird client filename. Many applications still look for `gds32.dll`. Firebird command-line tools like isql and gbak – which you can also run against the embedded server – want `fbclient.dll`. You can also make copies with both names.

2. Also copy `firebird.msg` and `ib_util.dll` to your application directory. Copy `aliases.conf` if your application uses aliases to connect. The configuration file `firebird.conf` is only needed if you want to change the Firebird root directory; this will be discussed later.

3. For Firebird 2 or higher, copy the `icu*.dll` libraries too.

4. From the `intl` and `udf` directories, copy whatever your application or databases may need to same-named folders under your application directory.

5. Now if you run your application it will use the embedded server DLL to connect to any local database you desire, *provided that the Windows user who runs the application has sufficient access rights to the database file(s) in question!* Any combination of user name and password is accepted, as long as neither is an empty string (a space is OK though).

The most important benefit of all this is that you can easily pack the Firebird Embedded files with your application and have them installed or unzipped at your users' computers without having to perform a separate Firebird install there (with all the additional worries of sufficient privileges, possible interference with already present Firebird or InterBase servers, etc. etc.). You can also have several embedded server versions (say 1.5.3 and 2.0) running on the same computer without the need for special configuration steps to keep them out of each other's way.

> **Note**
>
> Please note that, even though the security database is bypassed, SQL privileges – specified in the database itself – still apply: if you connect as Firebird user ZIGGY and ZIGGY doesn't have access to table STARDUST, you won't get into that table. This is not as worrying as it seems, because you can connect as any user you like, including SYSDBA, with a dummy password.

Placing the Firebird files elsewhere

By default, Firebird Embedded Server considers the folder that `fbembed.dll` lives in (under whatever name) as the Firebird root directory. In the setup described above, this is your application directory. You may want to place the Firebird files somewhere else, so as not to clutter your application directory. To do this, you must tell the server where to look for them.

Let's say you want the Firebird files to reside in a folder called `D:\FbEmbedded`:

1. Copy `firebird.conf` to your application directory and edit the *`RootDirectory`* parameter like this:

   ```
   RootDirectory = D:\FbEmbedded
   ```

   Alternatively, you may set the `FIREBIRD` environment variable to achieve the same. If both the configuration file parameter and the `FIREBIRD` envar are present, the latter takes precedence. **Warning:** the `FIREBIRD` environment variable will be picked up by every Firebird server on the system (embedded or not) at startup, and override their registry settings and configuration file parameters. So use it with care, if at all.

2. Copy/move the following items to `D:\FbEmbedded`:

   - `firebird.msg`
   - `aliases.conf` (if used)
   - the `intl` and `udf` subdirs plus contents (in so far as needed)

3. The following files however *must* be left in your application's directory:

   - Any and all copies/renames of `fbembed.dll`
   - `firebird.conf`
   - `ib_util.dll`
   - for Firebird 2 and up: the `icu*.dll` libraries

Embedded server on Linux?

The Linux Classic server comes with a client library called `libfbembed.so` which is used for local connections. This library contains a full Firebird engine, which explains why it's so much bigger than the Superserver client library. Local connections through this library are part of the user application process, not of a separate server process. Therefore, the user of the library must have filesystem-level access rights to the database – just as with Windows Embedded. So yes, this is a true embedded server.

There are also some differences. First, Linux Classic doesn't require an exclusive lock on the databases it opens. The database remains accessible from other clients. A further – very important – difference is that Linux Classic validates every login against the security database: no connections with phony user names or passwords here! Finally, you can't just ship `libfbembed.so` with your application and use it to connect to local databases. Under Linux, you always need a properly installed server, be it Classic or Super.

Application development

Once a database has been created and populated, its information can be accessed through a client application. Some applications – such as the Firebird isql tool, EMS SQL Manager, IB_SQL, IBAccess, Database Workbench, FlameRobin and IBOConsole – provide the capability to query data interactively and to create new metadata.

Any application developed as a user interface to one or more Firebird databases will use the SQL query language, both to define the sets of data that can be stored and to pass requests to the server about rows it wants to update, insert into or delete from. SQL statements also convey the values which the application wants to be applied to those rows.

Firebird implements a set of SQL syntaxes which have a high degree of compliance with the recognised SQL-92 and SQL-99 standards. The Firebird API provides complete structures for packaging SQL statements and the associated parameters, as well as for receiving the results.

Embedded SQL in Firebird applications

Firebird provides the capability to embed SQL statements in applications written in C/C++ and some other programming languages. The code is then passed through *gpre*, the pre-processor, which substitutes the embedded SQL statements with equivalent host language code that calls functions in Firebird's client API library. The gpre pre-processor generates a file that the host language compiler can compile.

A special, extra subset of SQL-like source commands is available for this style of application, which are pre-processed into internal macro calls to the API. Known as Embedded SQL (ESQL), it provides a simpler, high-level language syntax for the programmer, that gpre can interpret and re-code according to the more complex structure of the equivalent API calls.

> **Note**
>
> The *InterBase Embedded SQL Guide* (http://www.ibphoenix.com/downloads/60EmbedSQL.zip) provides extensive documentation on this subject.

Predefined vs. dynamic queries

Some queries have to be run in exactly the same form every time they are needed. Queries like this are good candidates for embedding in the host language and pre-processing by gpre. The pre-processor turns them into API function calls, giving a somewhat better performance than SQL that has to be interpreted at runtime.

But many applications need to build queries that are at least partially dependent on information provided by the user – freely entered in a text box, or selected from a list of options. This is called *Dynamic SQL* or *DSQL*; that is, SQL code whose form is not (or not exactly) known at design time. DSQL can be embedded and preprocessed too, but some additional requirements and restrictions apply. More on this – again – in the *InterBase Embedded SQL Guide*.

Delphi and C++ data access components provide properties and methods to analyse and parse DSQL request statements and manage the results passed back. Applications that use ODBC or other generic interfaces always work with DSQL statements, even if the user doesn't always see them. Query-by-example and other visual query tools for instance provide the user with a convenient, easy to use, and often “SQL-free” interface to extract, modify or delete data from the database. Yet the underlying code translates the user's input into DSQL statements, which are subsequently passed to the ODBC (or other) layer.

Component interfaces provide methods and properties for building and preparing SQL template statements, allowing you to use placeholders for value criteria. At run-time, the application supplies input parameters of the appropriate data type to complete the statement. Provision is made also for retrieving output parameters from statements that return results after they are executed.

> **Note**
>
> Of course the use of data access components isn't limited to dynamic SQL. You can also store static SQL strings – known at design time – in them.

RAD environments and component suites

With the rise of rapid application development (RAD) tools in the past decade, the encapsulation of the API functions in suites of components presents a variety of attractive application development options for Firebird developers.

The Borland Database Engine (BDE)

Borland markets “enterprise versions” of a number of integrated development tools – Delphi, Kylix, C++Builder, JBuilder and some older products – which can use the proprietary Borland Database Engine and native SQL Links InterBase drivers as a “black box” middleware layer to make InterBase and, latterly, Firebird databases behave like desktop databases. BDE version 5.2 and its associated InterBase driver, which first shipped with Delphi 6E, supports both Firebird and InterBase version 6, although it has known bugs affecting the Dialect 3 date and time data types.

Because the BDE's purpose is to surface a generic, database-vendor-independent, client-centered data access layer to the IDE tools, it flattens out the differences between a wide range of different database systems. Hence, it limits the capability of applications to exploit the best features of Firebird, particularly multiple transactions per connection, control of transaction aging and concurrency control.

The BDE can be useful where you need to develop an application that might be used with a choice of back-ends, of which Firebird is only one. Be warned however that people have reported problems with Firebird database access via the BDE, and these are likely to increase in number and severity as Firebird continues to move further away from InterBase.

SQLDirect

SQLDirect is a shareware, lightweight replacement for the BDE. It supports Firebird at least up to and including version 1.5. Visit http://www.sqldirect-soft.com for more information and free trial versions.

DBExpress and Datasnap

DBExpress and Datasnap were introduced in later versions of the Borland tools to provide alternative generic interfaces to databases. They replace the BDE by moving its functionality into expanded native drivers for supported databases. Like the BDE, they do not support multiple concurrent transactions. They are of especial use where a data interface is required that is independent of the idiosyncrasies of different database management systems. The InterBase native drivers should provide adequate support for Firebird databases where optimising client/server performance is not high among the objectives.

Direct-to-API Components

In response to the shortcomings of the BDE, a number of component suites have become available for Delphi and Borland C++Builder that bypass the BDE layer completely and encapsulate the Firebird API directly:

IBObjects
: IB Objects is a rich set of visual and non-visual database components that has been stable since 1997. It offers two BDE-free suites for data access; one compatible with the native Delphi and C++ Builder TDatasource

and visual controls, the other completely independent of the Delphi data access architecture and supplied with its own visual controls.

http://www.ibobjects.com

FIBPlus
FIBPlus is another well-known and stable suite for Delphi and BCB. Developed from Gregory Deatz's FreeIBComponents suite, it offers a connectivity based on TDataset. It doesn't include any visual components, but it can work perfectly together with the Borland visual database classes, as well as with third-party visual data-aware components.

http://www.devrace.com/en/fibplus/

ZeosLib
ZeosLib is a set of free, open-source database connectivity components for Delphi, FreePascal/Lazarus, Kylix and C++ Builder. It supports a number of database systems, including Firebird. Because the ZeosLib components are based on TDataset, you can use them together with the Borland visual database controls.

http://zeos.firmos.at/portal.php

IBX (InterBase Express)
IBX was also developed from the FreeIBComponents. Its TDataset-based data access components were purchased for developing as a proprietary product by Borland. Components encapsulating the new Services API were added. It was left unfinished in 1999. The IBX source code was opened under the InterBase Public License in 2000 and it continues to be developed as an open source project. Borland distributes versions of IBX with some Delphi, Kylix and C++ Builder versions.

> **Caution**
>
> Since InterBase and Firebird are diverging more and more, and Borland has (quite understandably) no intention to keep IBX Firebird-compatible, you should probably *not* use it with Firebird versions 1.5 and higher (although most features will still be supported).

UIB (Unified InterBase)
This is a set of non-visual components for Delphi, BCB, Kylix, Lazarus and FPC, supporting Firebird, InterBase and Yaffil. A ready-to-use SQL monitor (Windows only) is also available:

http://www.progdigy.com/modules.php?name=UIB

The UIB components are also contained in the JEDI Visual Component Library (JVCL):

http://homepages.borland.com/jedi/jvcl/

Both UIB and the JVCL are freely available open-source products.

Other connectivity components and drivers

Microsoft “open connectivity”

Third-party ODBC and OLE-DB drivers are available for Windows programmers using Microsoft and other vendors' programming tools. The Firebird ODBC driver development page is at http://www.firebirdtest.com/en/development-odbc-driver/.

Java

A pure Java Type 4 JDBC driver called Jaybird is developed within the Firebird project. Jaybird is compliant with both the new JCA standard for application server connections to enterprise information systems and the established JDBC standard.

> **Note**
>
> Firebird has abandoned re-implementation of the Borland InterClient and Interserver platform-independent client layers.

Official documentation and download links for Jaybird are at http://jaybirdwiki.firebirdsql.org.

.NET

The Firebird ADO.NET Data Provider, developed as a Firebird subproject, re-implements the client API functions in C#. Consequently, .NET developers only need the data provider to talk to a Firebird server; there's no need to install a regular Firebird client. Home page: http://www.firebirdtest.com/en/development-net-provider/.

API applications

The Firebird client program supports two discrete application programming interface (API) modules. The most important is the core API, through which all database work is performed. A much smaller API module, the Services API, provides functions for accessing various command-line and other utilities from application programs. Developers can write high-level programming or script language applications that populate the data structures and call the API functions directly.

The Firebird core API

Programmers who want to use the core API have to write code for allocating and populating the data structures that provide the communication layer between the client library and the server. Interactive SQL clients, component interfaces and embedded SQL “hide” these structures from the programmer by encapsulating them in their own higher level interfaces. Writing code that calls the API functions directly can be more powerful and flexible, with the following benefits:

- Memory allocation control
- No precompiling necessary

- Access to transaction handles and options
- Full access to error messages

Core API function categories

Based on their operation targets, we can divide the API functions into the following categories:

- Database connection control
- Transaction control
- Statement execution
- Blob functions
- Array functions
- Security functions
- Informational functions
- Type conversions

The Services API

The opened InterBase 6.0 code from which Firebird was developed surfaced for the first time an application programming interface (API) providing a function call interface to certain server activities such as backup/restore, statistics and user management. Many of these calls provide programming interfaces to the code in the command-line tools. A few lower-level server functions are included as well, some of which overlap functions already available in the core API.

> **Important**
>
> Before Firebird 1.5, the Services API was only available with Firebird Superserver. Support for the entire Services API in Classic Server versions was completed in Firebird 1.5.1.

Borland's InterBase Express (IBX) components include a subset – known as the Service components – encapsulating access to services API calls from some versions of their Delphi, Kylix and C++Builder development environments. Be aware however that IBX does not officially support Firebird. The higher your Firebird version, the more chance of incompatibilities and errors if you use IBX.

IBPP

IBPP is a C++ interface to the Firebird API. It is a class library written in “pure” C++ and hence not dependent on any specific programming environment, component set or even OS. You can use it anywhere you want Firebird connectivity in your C++ programs.

IBPP was created and is maintained independently of the Firebird project. It is available for free and comes with its own very liberal license. The IBPP home page is at http://www.ibpp.org.

Server-side programming

Among Firebird's powerful features for dynamic client/server application programming is its capability to precompile source code on the server, storing the object code right inside the database in most cases. Such proce-

dures and functions are executed completely on the server, optionally returning values or data sets to the client application. Firebird provides three styles of server-side programming capability: stored procedures, triggers and user-defined functions (UDFs).

Stored procedures

Firebird's procedural language (PSQL) implements extensions to its SQL language, providing conditional logic, flow control structures, exception handling (both built-in and user-defined), local variables, an event mechanism and the capability to accept input arguments of almost any type supported by Firebird. It implements a powerful flow control structure for processing cursors which can output a dataset directly to client memory without the need to create temporary tables. Such procedures are called from the client with a SELECT statement and are known to developers as *selectable stored procedures*. Procedures that don't return a dataset (although they may return result variables) are called *executable stored procedures*; they are called with EXECUTE PROCEDURE.

Stored procedures can call other stored procedures and can be recursive. All stored procedure execution, including selection of data sets from procedures and embedded calls to other procedures, is under the control of the single transaction that calls it. Accordingly, the work of a stored procedure call will be cancelled totally if the client rolls back the transaction.

Triggers

Triggers are special procedures created for specific tables, for automatic execution during the process of committing DML work to the server. Any table can have any number of triggers to be executed before or after inserts, updates and deletions. Execution order is determined by a position parameter in the trigger's declaration. Triggers have some PSQL extensions not available to regular stored procedures or to dynamic SQL, most notably the context variables OLD and NEW which, when prefixed to a column identifier, provide references to the existing and requested new values of the column. Triggers can call stored procedures, but not other triggers.

Work performed by triggers will be rolled back if the transaction that prompted them is rolled back.

PSQL limitations

Stored procedures and triggers can not start transactions, since they are under transaction control themselves.

PSQL does not allow the execution of DDL (Data Definition Language) statements: it is strictly meant to operate on *data*, not on the structure of your database. Although you can circumvent this limitation with the EXECUTE STATEMENT syntax introduced in Firebird 1.5, it is generally considered unwise to do so. (Just because we give you a spade, it doesn't mean that you have to dig your own grave.)

User-defined functions

By design, in order to preserve its small footprint, Firebird comes with a very modest arsenal of internally-defined (native) data transformation functions. Developers can write their own very precise functions in familiar host-language code such as C/C++, Pascal or Object Pascal to accept arguments and return a single result. Once an external function – UDF – is declared to a database, it becomes available as a valid SQL function to applications, stored procedures and triggers.

Firebird supplies two libraries of ready-to-use UDFs: `ib_udf` and `fbudf`. Firebird looks for UDF libraries in its own `UDF` subdirectory or in other directories specified in the Firebird configuration file. In Firebird 1.5 and upward this is done with the *`UDFAccess`* parameter; in earlier versions with *`external_function_directory`*.

> **Note**
>
> In versions prior to 1.5 the `fbudf` library is only available on Windows.

Multi-database applications

Firebird applications can work with several databases at the same time through the client library – something that not all relational database systems allow. Tables from separate databases can not be joined to return linked sets, but cursors can be used to combine information.

If consistency across database boundaries is required, Firebird can manage output sets from querying multiple databases inside a single transaction. Firebird implements automatic two-phase commit when data changes occur, to ensure that changes cannot be committed in one database if changes in another database, within the same transaction context, are rolled back or lost through a network failure.

Appendix A: Document history

The exact file history is recorded in the `manual` module in our CVS tree; see http://sourceforge.net/cvs/?group_id=9028

Revision History

0.0	2002	IBP	Written and published by IBPhoenix.
1.0	2005	IBP	Donated to Firebird Project by IBPhoenix.
2.0	12 Nov 2006	PV	*Preface* added. *About Firebird*: Updated to 1.5 and 2.0. Rewrote/corrected/extended a number of sections. Added section on Foundation. Introduced Embedded Server. Introduced aliases and (other) security features. Introduced nbackup. Introduced Services Manager + API. Added rows *Availability*, *Local access on Windows*, *Multiprocessor support*, *Services Manager + API*, *Guardian on Windows* and *Guardian on Linux* to the Classic-Super table. Added same-named sections to the Classic-Super discussion following the table. Explained bulkiness of Linux Classic client. Added note on AMD processors. *About Clients and Servers*: Updated to 1.5 and 2.0. Rewrote/corrected/extended a number of sections. Added sections on client filenames, multiple servers and server filenames. Added major section on embedded servers. Introduced SQLDirect. Warned against use of IBX with higher Firebird versions. Introduced UIB and JVCL. Introduced .NET data provider. Introduced IBPP. Explained the concept of executable stored procedures. Added section on PSQL limitations. *Document History* added. *License Notice* added.
2.0.1	31 Dec 2006	PV	*About this book :: More documentation*: Changed text and hyperlink. *About Firebird*: Changed title of memory requirements table. *About Clients and Servers*: Added ZeosLib to Direct-to-API Components section. Gave all tables and figures an ID to make their URLs persistent across builds.
2.0.2	16 Jul 2007	PV	*About this book :: Summary of features*: Gave table a "`keep-together=auto`" PI, necessary with FOP 0.93+ *About Clients and Servers :: Client and server combined: Firebird Embedded Server*: Added titleabbrev. Also corrected the first Note, which incorrectly stated that Fb2's `fbembed.dll` can't be used as a network client (altered 1st sentence, dropped last). Put "`localhost:`" in a `quote` in that same note. *License notice*: Updated copyright year.

Appendix B: License notice