



*MWA Software*

# **Firebird Server Management using IBX**

Issue 1.0,  
Doc Ref: MWA/IBX/SvrMgr  
9 February 2022

McCallum Whyman Associates Ltd

Email: [info@mccallumwhyman.com](mailto:info@mccallumwhyman.com), <http://www.mccallumwhyman.com>

## **COPYRIGHT**

The copyright in this work is vested in McCallum Whyman Associates Ltd. The contents of the document may be freely distributed and copied provided the source is correctly identified as this document.

© Copyright McCallum Whyman Associates Ltd (2018)  
trading as MWA Software.

## **Disclaimer**

Although our best efforts have been made to ensure that the information contained within is up-to-date and accurate, no warranty whatsoever is offered as to its correctness and readers are responsible for ensuring through testing or any other appropriate procedures that the information provided is correct and appropriate for the purpose for which it is used.

CONTENTS	Page
<b>1 INTRODUCTION.....</b>	<b>1</b>
1.1 REFERENCES.....	2
<b>2 FIREBIRD SECURITY.....</b>	<b>5</b>
2.1 OVERVIEW.....	5
2.2 RELIANCE ON FILE SYSTEM SECURITY.....	9
2.3 DATABASE ENCRYPTION.....	10
2.3.1 Server Side Database Encryption.....	10
2.3.2 Client Side Database Encryption.....	10
2.4 FIREBIRD USERS AND USER AUTHENTICATION.....	11
2.4.1 The Firebird Security Database.....	11
2.4.1.1 Security Database Configuration.....	12
2.4.1.2 Creating a Firebird Security Database.....	12
2.4.1.3 The Firebird 3 Default Security Database.....	12
2.4.1.4 Firebird 3 Alternative Security Databases.....	14
2.4.1.5 Alternative Security Databases and the Services API.....	15
2.4.1.6 Encrypting the Security Database.....	15
2.4.2 Authentication.....	16
2.4.2.1 Legacy Authentication.....	17
2.4.2.2 Secure Remote Password (SRP).....	17
2.4.2.3 Win_Sspi.....	21
2.4.2.4 SSL/TLS.....	22
2.4.2.5 Summary.....	22
2.4.3 User Management.....	23
2.4.3.1 Firebird 2.1 and Earlier.....	23
2.4.3.2 Firebird 2.5.....	23
2.4.3.2.1 CREATE USER.....	24
2.4.3.2.2 ALTER USER.....	24
2.4.3.2.3 DROP USER.....	24
2.4.3.3 Firebird 3.....	24
2.4.3.3.1 DDL Extensions.....	25
2.4.3.3.2 User Tags.....	25
2.4.4 Mapping Users between Security Contexts.....	25
2.5 ROLES.....	27
2.5.1 Creating a Role.....	27
2.5.2 Users and Roles.....	28
2.5.3 Assuming a Role.....	28
2.5.4 The RDB\$ADMIN role.....	28
2.5.4.1 Assuming the RDB\$ADMIN role when using the Services API.....	29
2.5.4.2 With Trusted Authentication.....	29
2.5.4.3 Use with the Security Database.....	29
2.5.4.4 RDB\$ADMIN and the Services API.....	29
2.5.5 The Trusted Role.....	30
2.6 ACCESS RIGHTS.....	30
2.6.1 Access Rights to Database Objects.....	30
2.6.1.1 Stored Procedures and Triggers.....	32
2.6.2 User Privileges for Metadata Changes.....	32
2.6.3 Stale Access Rights.....	33
2.7 SUMMARY.....	34
2.7.1 Issues.....	34
2.7.2 Requirements on a Database Management Tool.....	34
<b>3 USING THE IBX SERVICES API FOR DATABASE ADMINISTRATION.....</b>	<b>37</b>
3.1 RUNNING THE EXAMPLE.....	37
3.2 DATABASE BACKUP.....	38
3.3 DATABASE RESTORE.....	39
3.4 SERVER LOG.....	39
3.5 USER MANAGEMENT.....	40
3.6 THE “DATABASE” ACTIONS.....	41
3.6.1 Show Statistics.....	41
3.6.2 Validation.....	41

3.6.3	<i>Limbo Transaction Resolution</i>	41
3.6.4	<i>Database Sweep</i>	43
3.6.5	<i>Database Shutdown</i>	44
3.6.6	<i>Bringing a Database Backup Online</i>	44
3.7	USING AN ALTERNATIVE SECURITY DATABASE	45
3.8	HOW IT WORKS	45
3.8.1	<i>Services API Login</i>	47
3.8.2	<i>IBX for Lazarus and the Services API</i>	47
3.8.3	<i>IBX 2.2 and the Services API</i>	47
3.8.4	<i>User Management</i>	47
3.8.5	<i>Limbo Transactions</i>	48
3.8.6	<i>Using Alternative Security Databases</i>	48
3.9	SUMMARY	49
<b>4</b>	<b>THE DBADMIN TOOL</b>	<b>51</b>
4.1	RUNNING THE EXAMPLE	52
4.2	DATABASE PROPERTIES	52
4.2.1	<i>Database Backup</i>	54
4.2.2	<i>Database Restore</i>	54
4.3	THE FILES PAGE	55
4.3.1	<i>Adding a Secondary File</i>	55
4.3.2	<i>Shadow Sets</i>	56
4.3.2.1	<i>Adding a Shadow Set</i>	56
4.3.2.2	<i>Dropping a Shadow Set</i>	57
4.4	THE ATTACHMENTS PAGE	57
4.5	THE STATISTICS PAGE	57
4.6	SCHEMA PAGE	58
4.7	THE SERVER PAGE	58
4.8	THE USER MANAGER PAGE	58
4.9	THE ACCESS RIGHTS PAGE	60
4.9.1	<i>Stale Users</i>	62
4.10	THE AUTH MAPPINGS PAGE	62
4.11	THE DATABASE REPAIR PAGE	63
4.11.1	<i>Database Sweep</i>	63
4.11.2	<i>Online Validation</i>	63
4.11.3	<i>Database Validation</i>	64
4.11.4	<i>Kill Shadows</i>	65
4.12	THE LIMBO TRANSACTIONS PAGE	65
4.13	HOW IT WORKS	66
4.13.1	<i>Database Connections</i>	66
4.13.2	<i>Service API Login</i>	66
<b>APPENDIX A</b>	<b>POTENTIAL THIRD PARTY SECURITY PLUGINS</b>	<b>67</b>
A.1	USE OF SRP FOR DATABASE KEY MANAGEMENT	67
A.2	AN SSL/TLS PLUGIN FOR FIREBIRD	68
<b>NOTES</b>		<b>71</b>

# 1

## Introduction

This document is a supplement to the IBX For Lazarus User Guide and explores how *IBX for Lazarus* can be used to create applications that perform Firebird Server Management and Firebird User Management and which manage user access rights to Firebird Databases.

Server Management is primarily provided through the Services API and IBX provides a set of components on the “Firebird Admin” palette that may be used for different aspects of Server Management, including database backup/restore, statistics collection, database validation, Limbo Transaction resolution and User Management.

Prior to Firebird 3, there was a single (global) security database (containing user credentials) per server. Many access rights were implicit (e.g. creating tables or even whole databases) and the granting and revoking of access rights was performed using DDL statements.

Firebird 3 has improved upon this by:

- Permitting User Management to be performed using a combination of virtual tables and DDL Statements.
- Supporting alternative security databases on a per database basis.
- Allowing all access rights to be explicitly managed through DDL statements.

Use of the Services API is necessary for many server and database management tasks including database repair and maintenance, backup and restore. However, use of the Services API for User Management is now deprecated in Firebird 3 in favour of User Management via a database connection. For legacy support, the Services API is still available for User Management. However, it is limited to the global security database.

- Through the IBServices unit and the TIBSecurityService, IBX made available access to the Firebird Services API and hence could support User Management applications. This functionality continues to be available for legacy use.

- In IBX 2.1, the TIBUpdate component was introduced. This is intended to support dataset update using DDL statements. That is datasets generated from Firebird virtual tables and presented to the user using TIBQuery. Together they enable Firebird 3 style User Management through virtual tables and supporting DDL statements in a straightforward manner.

The *IBX for Lazarus* source code provides examples for both legacy and Firebird 3 User Management. This guide supports these examples and attempts to explain:

- How IBX may be used to support legacy User Management through the Services API, alongside other Server Management activities.
- How IBX may be used to support Firebird 3 User Management and the assignment of extended Access Rights

In order to understand what User Management actually means in Firebird and how it and other aspects of Server Management relate to Firebird Security, this guide starts with a review of Firebird Security.

## 1.1 References

1. IBX for Lazarus User Guide -MWA Software – Issue 1.5  
<https://mwasoftware.co.uk/downloads/send/5-ibx-current/147-ibx4lazarusguide>
2. Firebird 3.0.3 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf)
3. Firebird 2.0.7 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/html/rlsnotes207.html](https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes207.html)
4. Firebird 2.1.7 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/html/rlsnotes217.html](https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes217.html)
5. Firebird 2.5.8 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-2.5.8-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-2.5.8-ReleaseNotes.pdf)
6. Firebird 2.5 Language Reference Update  
<https://www.firebirdsql.org/refdocs/langrefupd25.html>
7. Firebird 1.5.6 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/html/rlsnotes15.html](https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes15.html)
8. Firebird 2.5 Language Reference  
[https://www.firebirdsql.org/file/documentation/reference\\_manuals/fblangref25-en/html/fblangref25.html](https://www.firebirdsql.org/file/documentation/reference_manuals/fblangref25-en/html/fblangref25.html)
9. US Department Of Defense Trusted Computer System Evaluation Criteria- DoD 5200.28-STD – December 1985  
<https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/dod85.pdf>
10. RFC 2945 The SRP Authentication and Key Exchange System, September 2000.

11. Wu, T., "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol", Submission to IEEE P1363.2 working group, October 2002, <http://srp.stanford.edu/srp6.ps>. See also: <http://srp.stanford.edu/design.html>
12. The Windows Security Support Provider Interface (SSPI) [https://msdn.microsoft.com/en-us/library/windows/desktop/aa380493\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa380493(v=vs.85).aspx)
13. RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication, November 2007.
14. ISO/IEC 27001 Information technology — Security techniques — Information security management systems — Requirements
15. How to use SRP in Openssl – <https://matthewarcus.wordpress.com/2014/05/10/srp-in-openssl/>
16. RFC 7568 Deprecating Secure Sockets Layer Version 3.0
17. RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2
18. Firebird Database Housekeeping Utility (gfix) - <https://www.firebirdsql.org/pdfmanual/html/gfix.html>
19. Firebird File and Metadata Security, December 2005 [https://firebirdsql.org/file/documentation/reference\\_manuals/user\\_manuals/html/fbmetasecur.html](https://firebirdsql.org/file/documentation/reference_manuals/user_manuals/html/fbmetasecur.html)
20. Research Results on SHA-1 Collisions <https://csrc.nist.gov/News/2017/Research-Results-on-SHA-1-Collisions>
21. NIST Special Publication 800-131A, Revision 1, Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths <http://dx.doi.org/10.6028/NIST.SP.800-131Ar1>





# 2

## Firebird Security

Before discussing how IBX for Lazarus can be used to manage Firebird users and perform other Database Administration Tasks, it is useful to have an appreciation of Firebird Security. The aim of this section is to present an understanding of how Firebird Databases are secured. At the time of writing there is no freely available guide to Firebird Security more recent than 2005 [19], and hence this section has had to collect together information from many sources including Release Notes for different versions of Firebird. The review is wider in scope than is strictly necessary for a day-to-day Database Administration Tool. However, it is also necessary to understand the context in which it is working and hence the scope of the review.

Notwithstanding the above, this review of Firebird Security is not intended to be definitive or complete but rather to give an understanding of the issues involved so that readers can read further and plan their own strategies for using Firebird Security functions. This is a complex subject and readers should assume that there are likely to be both errors and omissions in this review. When security mechanisms or other assumptions are being relied upon to protect sensitive data, they should always be subject to a program of analysis, verification and testing in order to demonstrate that those mechanisms perform their intended task and that any assumptions are valid and appropriate. Those with a clear need to develop an information security strategy may also wish to read ISO 27001 [14].

Note: throughout this section, endnotes are used to clarify various statements and to provide source references, in addition to direct references to documents. Direct references are in bold and enclosed in square brackets, while superscripts are used for endnotes. Quotations are highlighted in blue/grey.

### 2.1 Overview

In order to protect sensitive user data, Firebird implements a form of Discretionary Access Control (DAC) which itself has been “defined by the Trusted Computer System Evaluation Criteria [9] as:

A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is

capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).

These criteria require both

“an enforcement mechanism (e.g., self/group/public controls, access control lists)”

and require users both to identify themselves and to

“use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB<sup>1</sup> shall protect authentication data so that it cannot be accessed by any unauthorized user.”

Also relevant from the same source is the requirement for accountability.

Audit information must be selectively kept and protected so that actions affecting security can be traced to the responsible party. A trusted system must be able to record the occurrences of security-relevant events in an audit log. The capability to select the audit events to be recorded is necessary to minimize the expense of auditing and to allow efficient analysis. Audit data must be protected from modification and unauthorized destruction to permit detection and after-the-fact investigations of security violations.

### ***Protecting User Data***

Both the need to provide an “enforcement mechanism” and to “protect authentication” data demand at least either:

- a) that Firebird Servers rely upon physical access controls and the Host File System to protect local access to databases, or
- b) Firebird databases and authentication data are encrypted such that they are only available to authorised users.

That latter approach may be appropriate when there is a long term doubt over the security of the server while, the former, may be appropriate when the database server is housed in a secure environment managed by the data owner/controller.

Note: In either case, there is still a vulnerability to data loss whether through malicious action or system failure that needs to be countered through backups and other copies of the database being held at physically separate locations.

### ***Access Modes***

Firebird Servers provide two modes of user access to data:

- a) Embedded: where the server runs as part of the user process, and
- b) Remote: where some data communications mechanism is used to support data exchange between the server and a remote client. TCP/IP is the most common communications mechanism used; on Windows a local IPC based mechanism (xnet) is also available.

### ***Embedded Mode***

Embedded mode has been supported by Firebird since release 1.5 [7], although a similar capability could be found in some earlier versions of InterBase. In embedded mode, the server gives the user full access to any database that they can open and, hence, file system controls are the only mechanism used to protect access to user data in embedded mode. It is suitable for Personal Databases and is also necessary for some general administration tasks. On a general

purpose server, it can readily provide a means to circumvent data access controls if file permissions are not set appropriately.

## **Remote Access**

Remote Access is the typical means by which most users access Firebird Databases. Remote Access is not only useful in permitting access from remote systems, it also allows the Database Server itself to control access to individual objects within the database, in the line with the DAC; something that is just not enforceable in embedded mode. From Firebird 3 onwards, Firebird also supports “over the wire encryption” in order to protect remote access to user data where the communications path is vulnerable to attack<sup>2</sup>. In this context, Remote Access can also include local communication between a client and a server using IPC or loopback connections.

The Firebird Server provides remote access while running in SuperServer, SuperClassic or Classic mode. The release notes [2] describe each of these modes. However, for reasons due to conflicting locking strategies, the SuperServer mode should be avoided if concurrent use of embedded and remote access is deployed.

The Firebird Server also provides two independent APIs for Remote Access. These are:

- The Services API: used to perform administrative tasks including User Management, Backup and Restore, Statistics collection, etc.
- The Database API: used to support connections to databases using SQL.

Each is subject to a different user identification/authentication exchange (login) when access to the API is requested, and hence use separate connections to the server. Different access rights may be assigned to the same user when using each of the two APIs. Recent Firebird releases have also moved the User Management functions from the Services API to the Database API, whilst retaining some legacy User Management in the Services API..

## **Plugins**

Firebird 3 has introduced a “plugin” architecture where various functions are provided in external code libraries (DLLs in Windows, Shared Objects (SOs) in Unix). These are often used for security related activities such as User Management, User Authentication, Database Encryption, etc. Plugins can also be provided by third parties.

These code libraries must be protected from unauthorised modification by file system controls. Third party provided plugins should only be used from trusted sources.

## **Discretionary Access Rights**

In Firebird Remote Access Mode, Discretionary Access Rights are enforced such that:

- The objects to which access is controlled are items such as a Database Table and its data, and Stored Procedures.
- The subjects include Firebird “users” as well as “objects” taking on the role of a subject (e.g. Stored Procedures and Triggers).

- The groups correspond to SQL Roles. There is also special user/group called “PUBLIC” that all users implicitly belong to. The access rights given to “PUBLIC” correspond to world access.
- The access rights themselves include the rights to access and change database data and/or metadata<sup>3</sup>.
- All Firebird objects have an “owner” - a Firebird user – and which is able to grant subject access rights on the object. However, the permission to grant rights can also be passed on to other users.

### ***Firebird Users and Roles***

In Remote Access Mode, each Firebird user is identified through a unique login “UserName” (limited to 31 characters and constrained to the same syntax as an SQL Identifier) and who must be authenticated (logged on) when a connection is established to the database or to the Services API. Firebird also includes the notion of a “SuperUser”. That is a user that has full access rights to all objects. Firebird calls this user “SYSDBA”.

In line with SQL standards, a database designer can create named “user groups” (known as “roles” in SQL) and assign access rights to roles rather than to individual users. Users can be “granted access to a role” which permits them to logon under that role (by including a requested role name in their user credentials) and then assume all access rights granted to the role.

In general, this should be preferred to granting rights to individual users as it can greatly simplify database administration and reduces the risk of individual users keeping access rights that are inappropriate for their current use of the database.

Firebird 2.5 introduces a built-in role “RDB\$ADMIN” [5]. A user granted this role at the database level has SuperUser rights to all objects in that database. A user may also be granted this role at the server level and hence have the privilege to manage user accounts.<sup>4</sup>

### ***Granting Access Rights***

Access Rights on objects are granted/removed to/from subjects/groups using SQL Grant and Revoke statements on a per database basis. Access rights can only be granted when the user granting the right is (a) logged on to a database and (b) is the object owner or has been given permission by the object owner to grant the right, or is the SYSDBA<sup>5</sup>.

For example:

```
Grant Select on MyTable to Alice;
```

grants the “Select” access right to a user called “Alice” to a table called “MyTable”. The select access right allows Alice read access to the data stored in MyTable.

Note: Such statements cannot have placeholder parameters and while they can be executed by TIBSQL, the SQL statement has to be formatted and assigned to the component each time it is used.

Prior to Firebird 3, access rights to metadata were implicit and permissive<sup>6</sup>. Any Firebird user could create an object and become the object’s owner. In Firebird 3, the DAC has been extended to the metadata, with create/alter/drop access rights being available on metadata objects. There is also an explicit “create database” permission that a user must be given in order to be allowed to create a new database.

## User Logging and Accountability

Firebird does not automatically record user logins/logouts or any other auditable event in a log file as a standard feature. However, it is possible to create database connect and disconnect triggers, and triggers for other auditable events (such as table creation or row insert) as part of a database schema and record login/logout to a database table. Firebird 3 allows triggers for Create, Alter and Drop User events which can be used to provide accountability for some User Management actions.

## 2.2 Reliance on File System Security

As discussed above, when Firebird Databases are located on secure servers under the control of the data owner, there is no strong need to encrypt the database. Indeed, this was not even possible<sup>7</sup> prior to Firebird 3. However, even with physically secure servers, there is still a need to rely on file system security in order to protect user data.

Each Firebird Database comprises one or more files protected by the host file system. If the file system is compromised or its settings are weak enough to allow compromise then it is possible for an unauthorised user to:

- Copy a Database and read its data on another system with a Firebird Server.
- Modify or delete a database *in situ* (e.g. using embedded mode).

Assuming that the File System controls are themselves good, provide Discretionary Access Controls (for a multi-user environment), and are resistant to attack, these potential vulnerabilities can be mitigated by the following:

1. Creating an Operating System User Account dedicated to the Firebird Server (e.g. under Linux, this is typically the “no-login” account “firebird”), and that this account also “owns” all Firebird configuration files and databases. No normal user should have access to this account.
2. Running the Firebird Server under the “firebird” account.
3. Ensuring that the Firebird configuration files are writeable only by the “firebird” user (or an authorised System Administrator).
4. All Firebird databases and authentication data are readable and writeable only by the “firebird” user.
5. The File System settings for all directories in the file system path to each Firebird configuration file and database must not permit the creation, modification or deletion of files or sub-directories by unauthorised users.<sup>8</sup>

The Firebird configuration file “firebird.conf” has a “DatabaseAccess” setting that can limit where Firebird databases may be located. This may be used to limit these locations to those that the Database Administrator has ensured that are compliant with item 5 above. This avoids the risk of users creating databases in vulnerable locations.

Indeed, some may want to go further and set “DatabaseAccess = None” in the Firebird Configuration File. This means that the only databases that may be created or accessed are those which have aliases and explicit path names listed in the “databases.conf” file<sup>9</sup>. This can also be

used to hide database file system locations – as long as the “databases.conf” file is not readable by unauthorised users.

## 2.3 Database Encryption

### 2.3.1 Server Side Database Encryption

A Server side Database Encryption capability was introduced in Firebird 3 and is potentially useful in situations where a database cannot be fully protected from unauthorised access, such as on a “cloud server”<sup>10</sup>, or just as an extra level of security where sensitive data is concerned.

The “capability” is to support a “plugin” that may be used to encrypt or decrypt a database. There is no default database encryption plugin, although third party plugins are becoming available. The encryption strategy is understood as working at the page level with data, index and blob pages encrypted<sup>11</sup>.

A key issue with database encryption is key management and specifically how the encryption key is communicated to the plugin. It is clearly undesirable to keep the key in an unencrypted file on the server, as that would defeat the objective of protecting access to user data on a server where there was a significant risk of unauthorised access. Although a database encryption key could be saved in an encrypted form on the the server. The database encryption key must be communicated or decrypted when either:

- a) the user logs on, or
- b) by a System Administrator using some secure out-of-band channel (e.g. using SSH) prior to use by a database's users<sup>12</sup>.

In the former case, this demands that the client is able to authenticate the server, prior to passing the encryption key, and that the database encryption key is protected when it is communicated to the server (e.g. by over the wire encryption).

Note: if the client has no means of authenticating the server prior to passing it the encryption key, then an attacker may be able to masquerade as a genuine server and solicit the database encryption key from an unsuspecting client.

The latter case also requires that the System Administrator is able to authenticate the server. However, this case may also have the advantage that it could be reasonably assumed that a System Administrator will be more diligent in ensuring that all steps have been taken to authenticate the server and in protecting the database encryption key than would an ordinary user.

A “KeyHolder” plugin capability also exists to support database encryption key exchange from client to server, and this seems to be the steer<sup>13</sup> given by the Firebird 3 Release Notes. That is for the client to pass the database encryption key to the server when the user logs on. However, this does mean that the encryption key then has to be securely passed to each authorised client. In turn, this creates additional vulnerabilities: e.g. compromise of a single weakly protected client may be sufficient to allow an attacker access to the full database. It is also difficult to revoke access to a client already in possession of the key.

### 2.3.2 Client Side Database Encryption

Client side encryption does not appear to be part of the current architecture. Server side encryption puts obstacles in the way of an attacker and hence raises the cost of an attack but if the reason for the encryption is doubt over the security of the server then server side encryption can never offer a

perfect solution<sup>14</sup>. Client side encryption can avoid risks to sensitive data as a successful attack on the server is not sufficient in itself to reveal user data to attacker. However, client side encryption would normally be expected to work at the column level rather than the page level of server side encrypted. Protecting access to data using column level encryption alone may thus still reveal information about data relationships.

Arguably, client side encryption is outside the scope of the database engine. However, from the user's point of view it may need to be considered as part of a complete solution with both client and server side encryption working together to counter any perceived threats.

## 2.4 Firebird Users and User Authentication

A key requirement for Remote Access to Firebird Databases is that there must exist a means to identify and authenticate remote users.

Prior to Firebird 3, there was a single authentication mechanism based on a simple password exchange. This is now known as “legacy authentication”. Firebird 3 allows for multiple authentication plugins each providing a different means of authenticating users directly or indirectly using an external authentication server.

- A Database Administrator can select the plugin(s) that are permitted to authenticate users via the “AuthServer” setting in the “firebird.conf” file.
- The plugin(s) that a client may use to request authentication by are likewise enumerated in the “AuthClient” setting.

When user authentication is performed by the plugin itself, there also needs to be a means of securely storing user authentication data on the server. A common means of storing such data is provided by the Firebird Security Database.

### 2.4.1 The Firebird Security Database

Prior to Firebird 3, each Firebird Server maintained a single local security database – now known as the “default security database”. This security database is a normal Firebird Database and typically contained a single table called RDB\$USERS (or even just “USERS” in earlier versions). The Firebird Server (authentication plugin) reads the security database whenever it needs to authenticate a user login. Maintaining the integrity and security of the data held in the security database is thus critical to managing a secure Firebird Server.<sup>15</sup>

Prior to Firebird 2.0, the security database was directly maintained by the *gsec* utility and could be accessible to remote users. It was thus potentially available for read access by various users and hence could expose user credentials which, in turn, became vulnerable to a brute force attack on the password hashes stored in the database<sup>16</sup>. Firebird 2.0 changed *gsec* to use the Firebird Services API and hence permitted the security database to be hidden from user view, accessible only to the Firebird Server.<sup>17</sup>

In Firebird 3:

- the plugin architecture allows for multiple user authentication mechanisms and each can save their authentication data in their own tables in a Firebird Security Database.

- alternative security databases can also be defined on a per database level. That is the original Firebird Security Database is the default security database unless an alternative is defined for a given database.
- It is also possible for a user authentication plugin to save its authentication data in a user database<sup>18</sup>. This may well be desirable when the database is encrypted. The encryption mechanism can then be used to protect both the user data and the user authentication data.<sup>19</sup>

Firebird 3 has also restored the ability of users to establish a direct embedded connection to a security database (or even a remote connection if the server configuration is permissive enough). This restored ability is necessary to create alternative security databases, but needs to be managed with care if Firebird's previous vulnerabilities are not to be re-introduced. However, once a security database has been created (see 2.4.1.4), it can be hidden from view and only needs to be accessible to the Firebird Server.

#### **2.4.1.1 Security Database Configuration**

Prior to Firebird 3, the location of the security database was compiled into the server. In Firebird 3, the location is configurable.

The default configuration for the security database in Firebird 3 is split between “firebird.conf” and “databases.conf”:

- In “firebird.conf”, the “SecurityDatabase” setting is configured with the path to the “default security database”.
- In “databases.conf”. The alias “security.db” is assigned to the database and its local configuration settings include “RemoteAccess = false”. This setting prevents remote access. File system settings should also be set so as to limit access to authorised users only; only a System Administrator should have embedded access.

Note: The “RemoteAccess” setting is generally useful as a means of limiting database access, for any database, to embedded mode only. It should always be set to “false” for a security database other than when it is embedded in a user database.

#### **2.4.1.2 Creating a Firebird Security Database**

Prior to Firebird 3, there was only a single Firebird Security Database and this was normally provided as part of the initial set of files when the server was installed. The location differed between platforms and OS distributions but the installed file was the same and included the RSB\$USERS table with a single initial (super) user “SYSDBA” and a default password “masterkey”. It was up to the Database Administrator (DBA) to change the SYSDBA password to something less well known and the create other users.

#### **2.4.1.3 The Firebird 3 Default Security Database**

With Firebird 3, such a simple strategy is no longer possible given that many different plugins could be available for user authentication. Different installers may take different strategies. Some may continue to provide a prepared version of the default security database with the SYSDBA user and a default password and set up for the default authentication plugin.

However, some may do no more than install an empty security database and leave it to the DBA to select the user authentication mechanism and create the SYSDBA user with an appropriate password. In many ways, this is better than installing the security database with a very well known



password, and which includes the risk of being left in place if the DBA installs an alternative preferred user authentication plugin without fully disabling the default.

The “CREATE USER” DDL statement (introduced in Firebird 3) also makes it straightforward for the DBA to create the SYSDBA user even when the security database is empty. The process is described in the Firebird 3 release notes [2]:

Initialization is performed in embedded mode using the *isql* utility. For an embedded connection, an authentication password is not required and will be ignored if you provide one. An embedded connection will work fine with no login credentials and “log you in” using your host credentials if you omit a user name. However, even though the user name is not subject to authentication, creating or modifying anything in the existing security database requires that the user be SYSDBA; otherwise, *isql* will throw a privilege error for the CREATE USER request.

The SQL user management commands will work with any open database. Because the sample database `employee.fdb` is present in your installation and already aliased in `databases.conf`, it is convenient to use it for the user management task.

1. **Stop the Firebird server.** Firebird 3 caches connections to the security database aggressively. The presence of server connections may prevent *isql* from establishing an embedded connection.
2. In a suitable shell, start an *isql* interactive session, opening the employee database via its alias:
 

```
> isql -user sysdba employee
```
3. Create the SYSDBA user:
 

```
SQL> create user SYSDBA password 'SomethingCryptic';
SQL> commit;
SQL> quit;
```
4. To complete the initialization, start the Firebird server again. Now you will be able to perform a network login to databases, including the security database, using the password you assigned to SYSDBA.

If the SYSDBA already exists then “ALTER USER SYSDBA...” can be used instead to change the SYSDBA password. From Firebird 3.0.1 onwards the statement `CREATE OR ALTER USER SYSDBA PASSWORD <password>` can also be used to initialize an empty security database. [2]

After it has been created, File System permissions must be used to protect the security database from unwanted attention (hopefully the installer will have done this for you):

- On a POSIX system, the Firebird Server usually operates under the user “firebird” and the security database is read/write to the “firebird” user (and possibly group) only with no other users permitted read or write access. In the above, *isql* needs to be run as user *root* in order to access the security database. After initialising the SYSDBA user, ensure that security database is owned (or at least read/write) by user “firebird” and no other normal user can read it. A typical installation will show the user and group as “firebird” (both read/write) with no “world access”. The directory path should also not be writeable by any user other than root or firebird, otherwise a malicious user could replace the security database with a compromised version.
- On Windows, similar remarks apply. *isql* should be run by a System Administrator and the Firebird security database only readable/writeable by the Firebird Server or a System Administrator.

#### 2.4.1.4 Firebird 3 Alternative Security Databases

Firebird 3 can also assign an alternative security database on a per user database basis. There can be as many alternative security databases as needed and they may be shared between user databases and/or included within a user database. Each security database lists a set of users and their passwords and each may have a different SYSDBA password.

Alternative security databases are defined in the server's *database.conf* configuration file<sup>20</sup>. This replaced the *aliases.conf* file used prior to Firebird 3 and shares a similar syntax. In order to use an alternative security database, you must set up an alias for the database in the configuration file and then associate the alternative security database with the database alias. For example:

```
private.security.db = $(dir_secDb)/private.security.fdb
{
    RemoteAccess = false
}
employee = $(dir_sampleDb)/employee.fdb
{
    SecurityDatabase = private.security.db
}
```

assigns an alternative security database to the example employee database. The example locates the private security database in the same directory as the default security database<sup>21</sup>. The alternative security database is defined by an alias which also permits the prohibition of remote access to this database. Using an alias also simplifies administration when several databases share the same alternative security database. If the alias name is the same as the database's then the security database is embedded in the user database.

Note: there is a “chicken and egg” problem with the creation of an alternative security database. In the above example, if the employee database did not exist when the entry was created then it can only be created if the alternative security database already exists and contains a SYSDBA user. However, if this is the only database served by the security database then it is not possible to create the SYSDBA user (if one does not exist) unless the database exists (although it may be empty) and can be connected to by isql in embedded mode.

An alternative security database is created using isql in embedded mode e.g.:

```
isql22 -user SYSDBA
SQL>create database 'private.security.db';
```

The above simply creates a regular Firebird Database. It is empty with no tables defined as yet. There is nothing to indicate its purpose;

Note: the location in which the database is created should be protected and not writeable by another other than a System Administrator or the Firebird Server. isql will need to be run by a System Administrator (e.g. root on a Posix system) and it may be necessary, after it has been created, to change the security database's ownership and file system access permissions so that it is exclusively available to the Firebird Server.

However, while the above creates the database, there is no connection to it. In order to create any user, including the SYSDBA user, you have to connect to a database which uses the newly created security database. This means that the using database must exist before its security database can be initialised. For example, following on from the above isql session:

```
SQL>connect employee;
SQL>create user SYSDBA password 'some-obscure-password';
SQL>commit;
SQL>quit;
```

Note: in order to create the SYSDBA, you have to connect to the database in embedded mode. Once the SYSDBA user has been created (it is automatically the superuser), you can now connect remotely as SYSDBA and create the other users. Creating the SYSDBA user also creates the tables needed to support the chosen user authentication plugin.

The CREATE USER syntax also allows the user authentication plugin to be specified and this may determine which authentication tables are created and how the user credentials are saved. For example:

```
SQL>create user SYSDBA password 'some-obscure-password' USING PLUGIN SRP;
```

If no plugin is specified then the default User Manager plugin is used (see 2.4.3).

#### 2.4.1.5 Alternative Security Databases and the Services API

In Firebird 3 the Services API can still be used to manage users in the default security database – but not those in an alternative security database. However, it is still necessary for there to be some access from the Services API to databases that use an alternative security database. For example, when using the Services API for backup/restore.

In order to direct the Services API login to the appropriate security database, the Services Parameter Block (SPB) has a new item code: `isc_spb_expected_db`<sup>23</sup>. The value of this item is the name of a database which uses the alternative security database. The login will then be authenticated using the security database associated with the “expected db”.

Note: From IBX 2.2 onwards, the Services API components support a new login parameter (`expected_db`) which corresponds to the SPB `isc_spb_expected_db` item code.

User Mapping can also be used to provide access using the Services API to users (authenticated using the default security database) to databases that use an alternative security database (see 2.4.4).

#### 2.4.1.6 Encrypting the Security Database

Firebird 3 does not appear to support encryption of a standalone security database. The subject is not discussed in the release notes and, without a reference implementation for database encryption, it is difficult to test for such a capability. The assumption is that encryption of a standalone security database is not possible.

However, if the security database is embedded within a user database which is itself encrypted then the security database would also be encrypted. This could be desirable in environments where server security is not under the control of the data owner (e.g. on cloud servers).

- If an encryption plug supports provision of the encryption key via a secure out-of-band channel then it may be possible to support such a mode of operation (although such a capability still needs to be demonstrated). However, no such plugin is believed to exist that supports such a capability.

- If the key has to be provided by the client, this means that the database encryption key has to be provided to the server prior to user authentication, given that the user authentication tables are in the encrypted database. On the other hand, a client should only pass the database encryption key to the server after it has authenticated the server. Otherwise, an attacker could set up a “server masquerade” attack to solicit the database encryption key.

This implies that in this particular case, when the security database is embedded in an encrypted user database, there has to be a clear separation between client and server authentication, with server authentication taking place first, the encryption key passed to the server, and thus permitting client identification and authentication.

Note: the use of the term “identification”. By providing an encryption key, the client may have already be assumed to be an authorised user. However, which one? And which access rights should they be granted. That is why identification and authentication of that identity still need to take place.

This is a potential requirement here on the user authentication scheme and which needs to be returned to after reviewing the user authentication plugins provided with Firebird.

## 2.4.2 Authentication

Prior to Firebird 3, only a single user authentication mechanism was supported by the Firebird Server. Firebird 3 supports multiple user authentication mechanisms through its plugin architecture, including:

- **Legacy\_Auth**: user authentication is compatible with older versions of Firebird and provided for backwards compatibility with older clients. This uses a simple password hash, is described as very insecure when compared with later versions, and its use should be discontinued as soon as possible.
- **Srp**: an implementation of the Secure Remote Password (SRP) protocol [10].
- **Win\_Sspi**: an interface to the Windows Security Support Provider Interface [12], and is backwards compatible with Windows trusted authentication supported by Firebird 2.1 and 2.5.

Three parameters in the “firebird.conf” file configure the use of user authentication plugins:

- **AuthServer**: determines the plugins used by a Firebird Server for user authentication and the order in which they are tried.
- **AuthClient**: similarly determines the plugins used by a Firebird client to authenticate themselves to a server and the order in which they are tried.
- **UserManager**: is the plugin used for managing the security database. When more than one plugin is listed then the first is the default (used when no plugin is specified in the CREATE/ALTER/DROP USER DDL statements described below).

In order to provide continuous authentication throughout the lifetime of a connection, the authentication scheme should also lead to key agreement for encryption keys used for over-the-wire encryption of the connection, and/or a cryptographic data integrity check added to each message. Otherwise, communications are vulnerable to man-in-the-middle attacks that are transparent to the authentication phase but may then record and/or modify subsequent communication.

### 2.4.2.1 Legacy Authentication

Legacy Authentication is Firebird 2.x compatible and uses a SHA-1 based hash of the password, limited to first eight characters of the password given by the user. The authentication scheme is believed to be vulnerable to brute force attacks<sup>29</sup>.

### 2.4.2.2 Secure Remote Password (SRP)

From Firebird 3 onwards, a shortened implementation of the SRP-6a variant [11] of the Secure Remote Password (SRP) protocol [10] is provided as the default user authentication mechanism. As described in the RFC:

(SRP) is suitable for negotiating secure connections using a user-supplied password, while eliminating the security problems traditionally associated with reusable passwords. This system also performs a secure key exchange in the process of authentication, allowing security layers (privacy and/or integrity protection) to be enabled during the session. Trusted key servers and certificate infrastructures are not required, and clients are not required to store or manage any long-term keys. SRP offers both security and deployment advantages over existing challenge-response techniques, making it an ideal drop-in replacement where secure password authentication is needed.

In order to understand what SRP does and what its vulnerabilities may be, it is first necessary to look in detail at the algorithm and the underlying mathematics. Review of the source code indicates that the Firebird implementation deviates in three places from standard SRP and the following is based on a review of the source code.

#### **SRP Analysis**

SRP uses a derivative of the Diffie-Hellman key agreement scheme to generate a shared session key and its security depends upon the intractability of the Discrete Logarithm Problem<sup>24</sup>. For each user recorded in the security database, SRP stores the triplet:

{ <username>, <password verifier>, <salt> }

where the “salt” is some random number used to increase the entropy of the password verifier, and the password verifier is computed as:

$$x = H(\text{<salt>}, H(I, \text{'.'}, \text{<raw password>}))^{25}$$

$$\text{<password verifier>} = v = g^x \bmod N$$

H is a Secure Hash Algorithm and I is the username, and where '.' represents concatenation. Firebird uses the 160 bit SHA-1 as its Secure Hash Algorithm<sup>26</sup>. Note that the computation of the password verifier takes place in modulo number space (mod N) using some well known generator (g).

In the Firebird implementation:

- $g = 2$ , and
- $N$  = a fixed 1024 bit prime number (see source code file auth/srp.cpp)

Authentication proceeds by the client and server respectively passing to each other the user name (I) and salt (s). Each also generates a new random number, and computing from these the Diffie-Hellman exponentials A and B, respectively, which are also exchanged:

Client	Server
$a = \text{random}()$ $A = g^a \bmod N$  $I, A \rightarrow$	$b = \text{random}()$ lookup(s,v) using I and compute $B = (kv + g^b) \bmod N$ $\leftarrow s, B$

k is a Multiplier parameter ( $k = H(N, g)$  in SRP-6a. (Previously, this was  $k = 3$  for legacy SRP-6).

Whilst A and B are exchanged in clear, the intractability of the Discrete Logarithm Problem means that it can be assumed that it is not computationally feasible for an attacker to derive either a or b from observing the exchange of either A or B, respectively. Both sides then compute a shared secret (S) and a session key (K) by:

$p = \langle \text{raw password} \rangle$ $x = H(s, H(I, ':', p))$ $S = (B - kg^x)^{(a + ux)} \bmod N$ $K = H(S)$	$S = (Av^u)^b \bmod N$ $K = H(S)$
--	--------------------------------------

The variable  $u = H(A, B)$ .

Note: it can also be observed that u and k serve no purpose other than to increase entropy. Indeed, that is their purpose and to increase the algorithm's resistance to brute force attacks.

A final verification exchange<sup>27</sup> is used to demonstrate that both client and server have computed the same S (shared secret) and K (K is derivative from S). The shared session key (K) generated by SRP is used by Firebird for an ephemeral encryption key used for “over the wire encryption”.<sup>28</sup>

$M1 = H(H(N)^{H(g)}, H(I), s, A, B, K)$ $M1 \rightarrow$	Verify M1
---	-----------

By verifying M1, the server can assert that the client has authenticated itself to the server. SRP normally returns an  $M2 = H(A, M1, K)$  in order to complete the handshake and to confirm key agreement to the client. However, this step appears to be absent in the Firebird implementation and hence the observation that Firebird implements a shortened version of SRP.

In consequence, over-the-wire encryption becomes essential if the client is to have any confidence in that it is communicating with the correct server. This is because, the only way a client knows that it has logged on to the correct server is implicit in that the encrypted data exchanged can be decrypted using the agreed session key (K).

Notwithstanding the above, the Firebird implementation of SRP does provide user identification and authentication to a server, without exchanging the password or a simple hash based on the password.

This is because:

1. If an incorrect password was used by the client, it would generate a very different value of  $S$  compared with that generated with the correct password. A server can assert that in order to generate  $S$  and hence  $M1$  correctly, the user must have provided the correct password/username combination and is hence the identified user.
2. If an incorrect password verifier was used by the server, it would generate a very different value of  $S$  and  $K$  compared with that generated with the correct password. By verifying an  $M2$ , the client could have asserted that this must be the server that holds the user's authentication credentials and is hence the intended server. However, in the Firebird implementation this is only achieved implicitly through successful use of over-the-wire encryption.

We can verify that both client and server have computed the same  $S$  as follows:

On the Server Side;

$$\begin{aligned} S &= (Av^u)^b \bmod N \\ &= (g^a g^{xu})^b \bmod N \\ &= g^{ab} g^{xub} \bmod N \\ &= g^{(a+xu)b} \bmod N \end{aligned}$$

On the Client Side:

$$\begin{aligned} S &= (B - kg^x)^{(a+ux)} \bmod N \\ &= (kv + g^b - kg^x)^{(a+ux)} \bmod N \\ &= (kv + g^b - kv)^{(a+ux)} \bmod N \\ &= g^{b(a+ux)} \bmod N \end{aligned}$$

By substituting for  $A$  and  $B$  and  $v = g^x$ .

Given that exponentiation is commutative in modulo number space, we can observe that

$$g^{(a+xu)b} \bmod N = g^{b(a+ux)} \bmod N$$

and hence that both sides have computed the same  $S$ .

## Vulnerabilities

For SRP and like any authentication scheme, vulnerabilities may exist under the headings of client and/or server masquerade, replay, repudiation and man-in-the-middle attacks. When SRP is also used to generate a shared encryption key, it may also be vulnerable to threats that result from revealing this key.

1. The security of the exchange ultimately depends upon the user keeping their password secure and the server keeping the password verifiers secure, otherwise both client and/or server masquerades are possible.
  - An attacker that steals a user's password can clearly impersonate (masquerade as) that user.
  - An attacker who was able to obtain a copy of the salt, user name, password verifier triplets could implement a limited server masquerade even when over-the-wire encryption is used. For example, to solicit a database encryption key from a client. Without over-the-wire encryption, server masquerade is trivial given that there is no server identity information exchanged or even a proof that the server has access to the password verifiers.
2. Replay attacks are where an attacker records an authentication exchange and plays it back in an attempt to fool the server into believing that they are a genuine user (or vice versa). In SRP, these are unlikely to be feasible as long as both  $a$  and  $b$  are genuinely random. This



is because there is then a very low probability that the server will choose a value of  $b$  that was the same used for any authentication exchange that an attacker replayed.

3. If an attacker steals the list of password verifiers, can they use it to reveal user passwords?

If the password verifier was simply a SHA-1 hash of the username and password then the answer to this question would be probably be yes given that SHA-1 password hashes appear to be vulnerable to brute force attacks<sup>29</sup>. However, SRP password verifiers are exponentials i.e.  $v = g^x \bmod N$  where  $x$  is the hash of the password salt, the username and the password. There is thus the additional protection of the intractability of the discrete logarithm problem that renders the password verifiers highly resistant to brute force attacks. It is thus reasonable to answer "No" to this question<sup>30</sup>.

4. If an attacker steals the list of password verifiers, can they use it to masquerade as a genuine user?

The password verifier  $v$  is computed as ( $v = g^x \bmod N$ ) where  $x$  is the hash of the password salt, the username and the password. If it is possible to re-arrange the client side formula for computing  $S$  ( $S = (B - kg^x)^{(a + ux)} \bmod N$ ) such that all references to  $x$  are replaced with  $v$  then the answer would be yes. Note that it may be assumed to be computationally infeasible to compute  $x = \log_g v \bmod N$ , and hence such a simple substitution is not useful.

Inspection of the client side formula for computing  $S$  suggest that this is not possible. Expansion of the formula will lead to (e.g.) terms such as  $(g^x)^x$  and which cannot be substituted for  $v$  and evaluated without being able to compute  $x = \log_g v \bmod N$ , even though  $v (= g^x \bmod N)$  is known.

5. Are man-in-the middle attacks feasible, either with or without knowledge of the list of password verifiers?

Under a man-in-the-middle attack, the attacker masquerades as the server to the client and masquerades as the client to the server. The immediate motivation would be to control the value of  $S$  used for each communication path and hence the encryption key for over-the-wire encryption, so that the attacker could intercept, record and/or modify the subsequent communications. If the subsequent communications are not encrypted then the attacker is always able to be transparent during the authentication phase in order to record and/or modify the data exchanged, with or without control over  $S$ . Over-the-wire encryption is thus assumed when discussing protection against man-in-the-middle attacks.

As described above, without knowledge of the password or  $v$ , no client or server masquerades are possible. If the attacker knows  $v$ , then it can masquerade as the server but not as the client, given that the password verifiers are highly resistant to brute force attacks (item 3 above) and knowledge of the password verifier is not sufficient to masquerade as a user (item 4 above). Hence, it may be concluded that man-in-the middle attacks are not feasible even when the attacker knows  $v$ . Although there is one caveat to this assertion (see item 7 below).

6. The authentication exchange results in the shared secret ( $S$ ) and is hence symmetric in outcome. Hence, SRP does not offer a non-repudiable proof of identity. That is while the Firebird SRP implementation may be good enough to prove to a server that the client is who they claim to be, to be a non-repudiable proof, the server must be able to demonstrate to a third party that it was the "client" who logged in and that the server had not forged the proof. The server is clearly able to generate an M1 from the information it already has and by simply assuming a value for  $a$  (and hence  $A$ ). Hence, receipt of an apparently valid M1 is



not sufficient for a third party to accept that it was the result of a genuine login by the client rather than a server-side forgery.

SRP thus does not offer a non-repudiable proof of client identification, and may hence be deemed vulnerable to repudiation threats. This may be important in applications where users need to be fully accountable for their actions.

## 7. How seriously do SHA-1 problems affect the Firebird implementation?

M1 is probably where the issue arises, given that  $v$  has the additional protection of the discrete logarithm problem. M1 is a simple hash of information that an attacker can gain through monitoring the communication plus  $K$ . As M1 is used as a proof of identity incorporating a shared secret then it may be considered to be a simple type of digital signature<sup>31</sup>.

NIST has disallowed the use of SHA-1 for such a purpose [21] and this may be sufficient for some organisations to avoid the use of Firebird or, at least, Firebird with SRP user authentication. However, given that the shared secret is ephemeral and depends on the value of  $b$  randomly chosen by the server, it is unlikely that SHA-1 weaknesses could be exploited to impersonate a user.

On the other hand, there is a risk that SHA-1 weaknesses could reveal  $K$  (e.g. by speeding up a brute force attack using a similar strategy to those used to reveal passwords from SHA-1 hashes). This may hence allow an attacker to determine the encryption key used by Firebird's over-the-wire encryption and hence to eavesdrop on client/server communications. This is because  $K$  is the over-the-wire encryption key.

This may include revealing the encryption key used for database encryption, if it is provided by the client to the server over the otherwise encrypted communication. Furthermore, if the value of  $K$  could be revealed in real-time, then a specific example of a man-in-the-middle attack is possible whereby the attacker is transparent during authentication but is then able to intercept and modify otherwise encrypted communications.

Whether or not such attacks are viable is beyond the scope of this review. However, the risk of using a hash algorithm with known weaknesses is that such attacks may be possible.

Although the RFC describes SRP as a “cryptographically strong network authentication mechanism”, the form of authentication offered is often known as “weak authentication”<sup>32</sup>. This is because authentication is based on a single proof – knowledge of a plain text password of up to 20 bytes or so in length<sup>33</sup>. The “cryptographically strong” tag applies to the difficulty of an attacker determining a password from observing the communication, brute force, or even by obtaining a copy of the authentication data held in the security database.

To qualify as “strong authentication”, there needs to be a third party involved in the proof and/or some form of two factor authentication. The proof should also be non-repudiable.

### ***SRP and Database Encryption Key Management***

It can be observed that SRP requires access to the password verifiers before the server can authenticate itself (implicitly) to the client. As discussed in 2.4.1.6 above, it is thus not feasible to use SRP with a security database embedded in an encrypted database.

On the other hand, a simple extension of SRP could be used to simplify database encryption key management. This idea is explored further in Appendix A.1.

### 2.4.2.3 Win\_Sspi

Win\_Sspi is not a user authentication scheme in itself, rather an interface between Firebird and the Windows Security Support Provider Interface (SSPI) API [3]. As documented by Microsoft:

(The) Security Support Provider Interface (SSPI) allows an application to use various security models available on a computer or network without changing the interface to the security system. SSPI does not establish logon credentials because that is generally a privileged operation handled by the operating system.

A security support provider (SSP) is contained in a dynamic-link library (DLL) that implements SSPI by making one or more security packages available to applications. Each security package provides mappings between the SSPI function calls of an application and the functions of an actual security model. Security packages support security protocols such as Kerberos authentication and LAN Manager.

Win\_Sspi is only available on Windows Clients and Servers, and requires a common domain of trust. That is both client and server must share the same security support provider. Win\_Sspi does not provide a means to generate a shared session key and hence does not support the current Firebird “over the wire” encryption mechanism<sup>34</sup>.

When used, users are logged in to the Firebird database using their Windows logon user name. Win\_Sspi implements windows trusted authentication and is backward compatible with 2.1 and 2.5 clients and servers running on windows.

In Firebird 2.1<sup>35</sup> and 2.5, the “Authentication” parameter can be set to:

- Native: Firebird's own authentication method
- Trusted: Windows authentication is used (equivalent to WinSspi).

**Note:** If a local Administrator or a member of the built-in Domain Admins group connects to Firebird using trusted authentication, he/she will be connected as SYSDBA.

- Mixed: either of the above may be used – if a user and password are specified then “native” is implied.

In Firebird 2.5, automatic SYSDBA mapping was given more control when operating under “Trusted Authentication” and with a new DDL instruction:

```
ALTER ROLE RDB$ADMIN SET/DROP AUTO ADMIN MAPPING
```

That could configure this function on and off on a per server basis<sup>36</sup>. The RDB\$ADMIN role is discussed later in 2.5.4.

### 2.4.2.4 SSL/TLS

Firebird does not currently implement SSL/TLS. However, the plugin architecture would allow third party development and deployment of an SSL/TLS provider to replace the existing (builtin) “Remote” provider. Appendix A.2 explores how SSL/TLS the security benefits that could arise from an SSL/TLS provider.

SSL/TLS could provide strong authentication with a non-repudiable proof of client login.

### 2.4.2.5 Summary

User Authentication in Firebird 3 is clearly a major improvement on earlier versions. The Firebird SRP implementation offers both explicit user and implicit server authentication (when used with over-the-wire encryption) and is resistant to brute force attacks on the user password and to man-in-the-middle attacks. As long as both a user's password and the security database are secured and are not available to an attacker, it is arguably as good as needed for most Firebird deployments and where a non-repudiable proof of client login is not required. It should always be used with over-the-wire encryption.

The main caveat is the use of SHA-1 as its secure hash algorithm. This algorithm has known weaknesses and with SRP there is risk that such weaknesses could be used to reveal the session key used for over-the-wire encryption. It is not known whether these weaknesses can be exploited in such a way. However, it would seem prudent for Firebird to use an improved secure hash algorithm in the future.

The Win\_Sspi approach used to be considered more secure than Firebird native authentication which is true when only legacy authentication is available. It may still be more convenient for Windows users. However, as it does not appear to support “over the wire encryption”, database users that need to communicate with a server over unsecured data communications paths are likely to prefer SRP with “over the wire encryption” even when both ends are Windows systems.

SRP is the only authentication plugin that supports wire encryption and which provides a means to securely pass a database encryption key to the server. It also appears possible for a third party to extend SRP (see Appendix A.1) in order to “piggy-back” a database encryption key distribution scheme using public/private key encryption on to SRP by adding an extra payload to the M2 exchange.

TLS-SRP [13] is probably the next major step in improving Firebird user authentication (see Appendix A.2).

## 2.4.3 User Management

Win\_Sspi does not require the Firebird Server to keep track of users, as this is done externally by Windows security support providers. However, both Legacy\_Auth and Srp authentication plugins maintain user credentials in the Firebird security database. In Firebird 3, each provides a User Management “plugin” for this purposes. In early versions, the Legacy\_Auth User Manager was simply a built-in part of Firebird.

### 2.4.3.1 Firebird 2.1 and Earlier

Prior to Firebird 2.5, only the *gsec* utility and the Services API could be used to manage users. Either could be used to:

- List all users and their attributes
- Create a new user
- Modify an existing user's password or attributes
- Delete an existing user.

In these earlier Firebird versions, user attributes are limited: UID, GID, First Name, Middle Name and Last Name and are for information only. Each user is identified by a unique “User Name” and which is the key to the user's entry in the security database. A User Name once created cannot be modified<sup>37</sup>.

Only the SYSDBA user may perform User Management activities.

#### 2.4.3.2 Firebird 2.5

In addition to use of *gsec* and the Services API for User Management, Firebird 2.5 also introduced the CREATE/ALTER/DROP USER DDL statements [6]. These enable User Management from a Database Connection. The user must be logged in as SYSDBA or with the role RDB\$ADMIN<sup>38</sup> (see 2.5.4). However, even a normal user can use ALTER USER to change their own password.

Firebird 2.5 also allows the “RDB\$ADMIN” role to be granted to a normal user in respect of the security database itself. The user may then use CREATE/ALTER/DROP USER DDL statements or the Services API to perform User Management in addition to the SYSDBA.

The new DDL statements are defined as:

##### 2.4.3.2.1 CREATE USER

Description: Creates a Firebird user account.

Syntax:

```
CREATE USER username PASSWORD 'password'
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname']
  [GRANT ADMIN ROLE]
```

GRANT ADMIN ROLE gives the new user the RDB\$ADMIN role in the security database. This allows them to manage user accounts, but doesn't give them any special privileges in regular databases.

##### 2.4.3.2.2 ALTER USER

Description: Alters details of a Firebird user account. This is the only account management statement that can also be used by non-privileged users, in order to change their own account details.

Syntax:

```
ALTER USER username
  [PASSWORD 'password']
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname']
  [{GRANT|REVOKE} ADMIN ROLE]

-- At least one of the optional parameters must be present.
-- GRANT/REVOKE ADMIN ROLE is reserved to privileged users.
```

##### 2.4.3.2.3 DROP USER

Description: Removes a Firebird user account.

Syntax:

```
DROP USER username
```

### 2.4.3.3 Firebird 3

The weakness in Firebird 2.5, from the point of view of User Management from a database connection is that there was no way to list the existing users. The Services API has to be used for this purpose. With the introduction of alternative security databases and the restriction of the Services API to the default security database, a new mechanism was needed to list the users. Use of the Service API for User Management is deprecated in Firebird 3<sup>39</sup>.

The SEC\$USERS virtual table was introduced in Firebird 3<sup>40</sup> and lists the users, and their attributes, in the security database associated with the database. If the user is logged in as SYSDBA or with the RDB\$ADMIN role (security database) then all users are listed. Otherwise, the list is restricted to the current user. The list is also restricted to the users managed by each of the User Managers listed in the “firebird.conf” configuration file.

Note: the user list can still be displayed using the services API. However, the list is limited to users managed using the default user manager only.

#### 2.4.3.3.1 DDL Extensions

Firebird 3 has also extended<sup>41</sup> the CREATE/ALTER/DROP USER syntax introduced in Firebird 2.5:

```
CREATE USER username [ options_list ] [ TAGS ( tag [, tag [, tag ...]] ) ]
ALTER USER username [ SET ] [ options_list ] [ TAGS ( tag [, tag [, tag ...]]
) ]
ALTER CURRENT USER [ SET ] [ options_list ] [ TAGS ( tag [, tag [,
tag ...]] ) ]
CREATE OR ALTER USER username [ SET ] [ options ] [ TAGS ( tag [, tag [,
tag ...]] ) ]
DROP USER username [ USING PLUGIN plugin_name ]
```

OPTIONS is a (possibly empty) list with the following options:

```
PASSWORD 'password'
FIRSTNAME 'string value'
MIDDLENAME 'string value'
LASTNAME 'string value'
ACTIVE
INACTIVE
USING PLUGIN plugin_name
```

Each TAG may have one of two forms:

```
TAGNAME = 'string value'
```

or the DROP TAGNAME tag form to remove a user-defined attribute entirely:

```
DROP TAGNAME
```

The Active/Inactive option allows a user to be disabled rather than deleted. The “USING PLUGIN” clause allows multiple authentication plugins to be used for the same security database. The name referred to is the User Manager plugin name and not the authentication plugin name.

Note: For Srp, both the User Manager and the authentication plugin have the same name “Srp”. For Legacy Authentication the names are respectively: Legacy\_UserManager and Legacy\_Auth.

#### 2.4.3.3.2 User Tags

User tags are additional attributes in “key=value” format that are available for use for unspecified purposes. They are managed using the CREATE/ALTER USER DDL statements and those specified for a given user can be read using the SEC\$USER\_ATTRIBUTES virtual table.

**Note:** The legacy user attributes UID and GID are copied to a Firebird 3 security database as tags when a Firebird 2 security database is upgraded<sup>42</sup>.

### 2.4.4 Mapping Users between Security Contexts

Firebird 3 introduces the concept of “mapping rules”. These have two purposes:

- a) To manage the mapping between users identified and authenticated by an external user authentication mechanism (e.g. a Windows Security Support Provider) into Firebird users and roles, and
- b) To allow users identified and authenticated using one user authentication plugin/security database to be mapped to users and roles in a database configured to use a different user authentication plugin/security database.

In both cases, the mapping takes place after authentication. Thus, it is not possible to (e.g.) set up a mapping that allows user authentication, at the point where a user connects to a database, to take place using a different user authentication plugin/security database from that defined for the database.

- In case (a) above, a user authenticated using an external user authentication mechanism (e.g. Windows SSPI) can be mapped so that they can be logged in seamlessly to databases and services without a further authentication step.
- In case (b) above, such mappings are limited to using the DML “EXECUTE STATEMENT ON EXTERNAL DATA SOURCE<sup>43</sup>” and the Services API and do not work with connections to databases.<sup>44</sup> This applies (e.g.) to mapping rules that allow a user (e.g. SYSDBA) authenticated in (e.g. the default security database) to be mapped to the SYSDBA user in a database using an alternative security database. Such a capability may be used to simplify overall database administration with a unified SYSDBA login – but for Services only.

Mapping rules are created by DDL statements executed in the database to which they apply. They are defined in [2] as:

```
{CREATE | ALTER | CREATE OR ALTER} [GLOBAL] MAPPING name
  USING {
    PLUGIN name [IN database] | ANY PLUGIN [IN database | SERVERWIDE] |
    MAPPING [IN database] | '*' [IN database]}
  FROM {ANY type | type name}
  TO {USER | ROLE} [name]
  --
DROP [GLOBAL] MAPPING name
```

- A Global Mapping applies to all databases that use the current security database. A local mapping applies only to the current database.
- The Using clause is used to identify the source of the mapping defined as an authentication method (plugin) and the security database in which the source of the mapping is defined. The “DATABASE” must conform with syntax rules for an SQL identifier.

Note: 'security.db' is conventionally defined in the Firebird databases.conf file to identify the default security database. It must be enclosed in double quotes, (e.g.) in the following example, as it includes the '.' character.

- The From clause identifies the type and name of the subject that is being mapped into the local database. The “type” is a subject defined by the plugin (with SRP “user” is a type name)
- The To clause identifies what is being mapped to. Either a user or a role.

An example of a mapping rule is:

```
CREATE MAPPING DEF_SYSDBA
  USING PLUGIN SRP IN "security.db"
  FROM USER SYSDBA
  TO USER;
```

which should allow the server's SYSDBA (authenticated using the default security database) to access the current database using the Services API (assuming that the database is using a non-default security database). Alternatively,

```
CREATE GLOBAL MAPPING TRUSTED_AUTH
  USING PLUGIN WIN_SSPI
  FROM ANY USER
  TO USER;
```

enables the use of Windows trusted authentication in all databases that use the current security database, while

```
CREATE MAPPING WIN_ADMINS
  USING PLUGIN WIN_SSPI
  FROM Predefined_Group
  DOMAIN_ANY_RID_ADMINS
  TO ROLE RDB$ADMIN;
```

enables SYSDBA-like access for windows administrators in current database.

Note: The group DOMAIN\_ANY\_RID\_ADMINS does not exist in Windows, but is instead added by the win\_ssapi plug-in to provide exact backwards compatibility.

## 2.5 Roles

Firebird implements “Roles” as specified in the SQL standard, where a role may be considered to be a named user group. Access rights can be assigned (granted) to roles in the same way as they are assigned to users (see 2.6). A user can be allowed to use (be granted) many different roles and may assume<sup>45</sup> a role that they are permitted to use either when they login or, from Firebird 3 onwards, at any time (see 2.5.3) during a database connection.

Once a user assumes a role they are implicitly granted all access rights assigned to the role in addition to any access rights they are assigned individually.

Roles can be viewed as a means of greatly simplifying user administration. A Database Architect may determine the different types of user that may access the database and the needs of each such user group. A role should be created for each such user group and appropriate access rights assigned to the role.

When a user is given access to a database, rather than having to work out which tables, views and stored procedures they need to access, all the database administrator need to do is to identify the most appropriate role(s) for them and grant them use of those roles.

### 2.5.1 Creating a Role

The creation of role objects should properly be considered as part of the database schema definition rather than a day to day Database Administration task. A role<sup>46</sup> is created using the DDL Statement:

```
Create Role <rolename>;
```

and deleted using:

```
Drop Role <roleName>;
```

### 2.5.2 Users and Roles

A user is granted use of a role using the statement:

```
Grant <rolename> to <username>;
```

and the use of a role revoked using the statement:

```
Revoke <rolename> from <username>;
```

Note: revoking the granted roles from a given user tidily removes their access rights to the objects in the database. However, this only affects the current database and not any backups. This is because the privilege to use a given role is held in the database itself. One of the tasks of Database Administrator is to keep track of when privileges are revoked so that should it be necessary to restore a database backup, all rights revoked since that backup are, once again, revoked.

### 2.5.3 Assuming a Role

Prior to Firebird 3, it was only possible to assume a role when a user logged into a database. Firebird 3 introduces the SET ROLE statement which allows a user to assume, post login, a role that has been previously granted to them. The statement syntax is:

```
SET ROLE <rolename>
```

Once a user has assumed a role, the CURRENT\_ROLE SQL variable is set to the rolename and the access rights assigned to the role are given to the user until they log out or change the current role.

### 2.5.4 The RDB\$ADMIN role

The RDB\$ADMIN role was introduced in Firebird 2.5 [5] in order to enable the transfer of

“SYSDBA privileges to another user. Any user, when granted the role in a particular database, acquires SYSDBA-like rights when attaching to that database with the RDB\$ADMIN role specified.”

The RDB\$ADMIN role does not have to be created and may be considered as being “built-in”. It is granted and revoked in the same way as any other role. Only the SYSDBA or a user assuming the RDB\$ADMIN role can grant or revoke this role.



- Granting a user the RDB\$ADMIN role for a given database allows them full SYSDBA access for that database e.g. to take backups, take the database offline, put it back online, etc. However, it does not allow them to manage other users.
- Granting a user the RDB\$ADMIN role in the security database additionally allows them the right to manage users in the security database. However, it does not permit them other SYSDBA rights (e.g. database backup), if they do not also have the RDB\$ADMIN role for the database.

A user granted the RDB\$ADMIN role in the security database only can only manage users using the Services API. In Firebird 3, they cannot use the SEC\$USERS table to list other users. They need to be granted the RDB\$ADMIN role in the current database before they can do this.

As with any other role, the user must assume the RDB\$ADMIN role when they log on or by using the “SET ROLE RDB\$ADMIN” statement.

#### 2.5.4.1 Assuming the RDB\$ADMIN role when using the Services API

In order to assume the RDB\$ADMIN role when logging into the services API, it has to be requested using the `isc_spd_sql_role_name` attribute in the Service Parameter Block.

**Note:** in IBX this is achieved by providing the line “`sql_role_name=RDB$ADMIN`” in the Params property.

#### 2.5.4.2 With Trusted Authentication

Firebird 2.5 also allowed Windows Administrators logged in using trusted authentication (see 2.4.2.3) to automatically be granted the RDB\$ADMIN role i.e.

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING;  
ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING;
```

are used to respectively to enable the capability and to disable it at the server level.

**Note:** in Firebird 2.1, the RDB\$ADMIN role did not exist and Windows Administrators logged in using trusted authentication automatically become the SYSDBA.

Firebird 3 has superseded auto admin mapping for Windows Administrators with the more general concept of user mappings (see 2.4.4).

#### 2.5.4.3 Use with the Security Database

Firebird 2.5 also allowed the RDB\$ADMIN role to extend to the security database – although this is more like an administrator privilege rather than a role and is granted/revoked using an entirely different syntax. That is using the CREATE/ALTER/USER statement (see 2.4.3.2).

When a user has been granted the “ADMIN ROLE” using (e.g.) an ALTER USER statement, they may use either the Services API or the CREATE/ALTER/USER statements to manage users.

The capability remains unchanged in Firebird 3.

#### 2.5.4.4 RDB\$ADMIN and the Services API

As described in the “Other Services API Additions” section of [8], the Services API was also extended by Firebird 2.5 to:

- a) Allow the auto mapping of the RDB\$ADMIN role to be enabled/disabled through the Services API, and
- b) Allow the SYSDBA to grant/revoke of the security database admin privilege to a user.

### 2.5.5 The Trusted Role

“SET TRUSTED ROLE” is an SQL statement which is used to set the current role based on a local or global mapping rule. When the statement is executed and a local or global mapping is found that explicitly or implicitly maps the current user to a role, then the user assumes the role and CURRENT\_ROLE is set to the selected role name.

Some security administrators may not be happy with potential lack of predictability with the the result of this statement. However, there appears to obvious way to disable it other than by not using mapping rules.

## 2.6 Access Rights

In embedded mode, the Firebird user can do anything they like to a database to which the file system allows them access. However, in Remote Access mode, Discretionary Access Controls are in force controlling access to database objects, including the database itself.

- Each Firebird object (e.g. database, table, view, store procedure, etc.) has an owner identified by their username, who is also the object's creator. An object's owner implicitly has full access to the object.
- An object owner or an administrator (SYSDBA or a user who has assumed the RDB\$ADMIN role) can grant (or revoke) access rights to (or from) an object including the right to grant/revoke access to/from other users.
- Access Rights include permissions to read, update, insert and delete rows in table, or to execute a stored procedure.
- From Firebird 3 onwards, metadata changes are also subject to privileges, e.g to create, alter or drop an object.

The creation of roles and the granting of access rights to roles should be considered to be part of the database schema and maintained as such. The granting of roles to users may then be considered as part of the day to day administration of the database. The granting of per user access rights to database objects should be avoided as this can be difficult to track and mange.

### 2.6.1 Access Rights to Database Objects

The general form of the SQL Grant statement is<sup>47</sup>:

```
GRANT
  <privileges> ON [TABLE] {tablename | viewname}
  | EXECUTE ON PROCEDURE procname
  | USAGE ON <object_type>
  }
TO <grantee_list>
  [WITH GRANT OPTION] [{GRANTED BY | AS} [USER] grantor];

GRANT <role_granted>
TO <role_grantee_list> [WITH ADMIN OPTION]
```

```

[ { GRANTED BY | AS } [ USER ] grantor ]

<privileges> ::= ALL [ PRIVILEGES ] | <privilege_list>

<privilege_list> ::= { <privilege> [, <privilege> [, ... ] ] }

<privilege> ::=
    SELECT |
    DELETE |
    INSERT |
    UPDATE [( col [, col [, ...] ] ) ] |
    REFERENCES ( col [, ...] )

<grantee_list> ::= { <grantee> [, <grantee> [, ...] ] }

<grantee> ::=
    [ USER ] username | [ ROLE ] rolename | GROUP Unix_group
    | PROCEDURE procname | TRIGGER trigrname | VIEW viewname | PUBLIC

<role_granted> ::= rolename [, rolename ...]

<role_grantee_list> ::= [ USER ] <role_grantee> [, [ USER ] <role_grantee> [, ...] ]

<role_grantee> ::= { username | PUBLIC }

<object_type> ::= { DOMAIN | EXCEPTION | GENERATOR | SEQUENCE | CHARACTER SET
    | COLLATION }

```

Different sets of privileges can apply to different SQL objects. The semantic of each privilege is well defined in SQL. In the above, the “object” of the access right is the database object that is to the right of the “ON” clause, while the subject is known as the “grantee”.

- When the “WITH GRANT OPTION” clause is present, the subject is given the right to grant the privilege to other users.
- The privilege is usually granted by the owner's object. However, it can be recorded as having been granted by another user – who has already been granted the right to grant the privilege. This is performed by the “GRANTED BY” clause.
- The “WITH ADMIN OPTION” applies to the granting of roles to users and permits the user to grant the role to another user.

The individual privileges are:

Select	The subject can retrieve all rows from the object (Table or View)
Delete	The subject can delete rows from the object (Table or View)
Update	The subject can modify the current value in one or more columns in the object (Table or View). This privilege can apply to all columns or just those selected.
References	This privilege allows the subject to add a Foreign Key reference from a table they own to one or more (or any) columns in the object table. The

	<p>reference cannot be added unless the privilege exists.</p> <p>Note: if the same user owns both table then there is no need to grant the privilege as the user has full access rights on both tables.</p> <p>A user who updates (the referencing) column in the subject table must also have been granted the REFERENCES privilege on the object table (e.g. through cascaded Foreign key updates).</p>
Execute	This privilege allows the subject to execute the specified stored procedure. This includes procedures that return datasets and are hence used in select statements.
Usage	Permits the use of an object such as an exception, generator, character set or collation.

### 2.6.1.1 Stored Procedures and Triggers

Stored Procedures and Triggers can also be granted access rights to database objects. When executed a Stored Procedure or Trigger executes with access rights inherited from the current user. However, there may be situations where the procedure or trigger is performing controlled actions on the user's behalf and it is appropriate to grant them access rights beyond those that are granted to the current user. SQL thus allows access rights to be granted to a stored procedure or trigger in the same way as they are granted to a normal user.

Some may view it as good practice to always explicitly grant a stored procedure or trigger all the access rights they need to perform their task rather than just assuming that the required access rights have been granted to the current user. Bugs due to a stored procedure or trigger not having sufficient access rights can be difficult to track down, especially in nested levels of calls.

### 2.6.2 User Privileges for Metadata Changes

In earlier versions of Firebird, any Firebird user could create a database or an object in a database, while only the object owner (or an administrator) could alter or drop an object. In Firebird 3, the database Metadata is also subject to access rights, permitting the right to create, alter or drop an object (including a database) to be individually granted<sup>48</sup>.

The syntax of metadata privilege grant/revoke statements is:

#### Granting metadata privileges:

```
GRANT CREATE <object-type>
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
GRANT ALTER ANY <object-type>
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
GRANT DROP ANY <object-type>
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
```

#### Revoking metadata privileges:

```
REVOKE [GRANT OPTION FOR] CREATE <object-type>
  FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] ALTER ANY <object-type>
```

```

FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] DROP ANY <object-type>
FROM [USER | ROLE] <user-name> | <role-name>;

```

**Special form for database access:**

```

GRANT CREATE DATABASE TO [USER | ROLE] <user-name> | <role-name>;
GRANT ALTER DATABASE
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
GRANT DROP DATABASE
  TO [USER | ROLE] <user-name> | <role-name> [WITH GRANT OPTION];
REVOKE CREATE DATABASE FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] ALTER DATABASE
  FROM [USER | ROLE] <user-name> | <role-name>;
REVOKE [GRANT OPTION FOR] DROP DATABASE
  FROM [USER | ROLE] <user-name> | <role-name>;

```

An <object-type> can be any of the following:

- CHARACTER SET
- COLLATION
- DOMAIN
- EXCEPTION
- FILTER
- FUNCTION
- GENERATOR
- PACKAGE
- PROCEDURE
- ROLE
- SEQUENCE
- TABLE
- VIEW

### 2.6.3 Stale Access Rights

Access rights assigned to a user name do not require the user to exist when they are granted and persist after a user has been deleted. The only check made is whether the current user has the required access rights to perform an operation on a database object and the check is a simple name match against the user names to which access rights have been assigned. A similar name match occurs when a user assumes a role.

**Note:** Granting an access right to a non-existent role fails with an error.

For this reason it is possible for “stale access rights” to exist in a database for which there is no matching user in the applicable security database. This may have occurred because:

- a) A simple typing error resulting in an incorrect and non-existent user name when an access right is assigned to a user.
- b) The user has been deleted from the database without revoking their access rights.
- c) A backup is restored which includes access rights to a user that had previously been deleted, even when their access rights had been revoked in what was then the current database. There is no option in the *gbak* utility to ignore access rights granted to non-existent users on database restore.

The risk resulting from stale access rights is that a System Administrator may, at some later date, create a new and entirely different user with the same user name who then has unintended access to database objects.

This problem has to be mitigated procedurally by a Database Administrator. Strategies include:

- From Firebird 3, it is possible to make a user “inactive” and hence unable to log in. Rather than deleting a user this can be used to avoid creating a new user with the same name as a previously deleted user.
- Firebird 3 also introduces the SQL statements:

```
REVOKE ALL ON ALL FROM [USER] username  
REVOKE ALL ON ALL FROM [ROLE] rolename
```

These can be usefully used to ensure that all access rights have been removed from a given user. A record should also be kept of deleted users and the above executed for each deleted user when a backup is restored.

- Never assign access rights to individual users. Only assign access rights to roles, and roles to users. Revoking a role from a user is then a single (revoke) command. Prior to Firebird 3, revoking all access rights assigned to an individual user would have required a sweep of the user privileges system table to identify all such access rights.

## 2.7 Summary

A clear conclusion from this review is just how many improvements to security there have been in Firebird 3. User Authentication is clearly far superior. Other features such as “over the wire encryption”, and access rights to metadata objects are also very significant improvements. Similarly, other, apparently minor changes such “REVOKE ALL ON ALL..” are also very important when it comes to managing a secure database.

### 2.7.1 Issues

On the other hand:

- The use of SHA-1 as the secure hash algorithm was unfortunate given that it is now deprecated in the wider security community. Potential issues have been discussed above. Hopefully this will be changed to a more secure hash algorithm in future releases.
- Database encryption is still a developing area for Firebird. In the standard release, the hooks exist, but require third parties to step forward with products. There is no default reference implementation. It is also unclear as to whether it is possible to encrypt the security database. SRP has vulnerabilities associated with theft on an unencrypted security database and security database encryption would be a useful countermeasure.

On the other hand, the plugin architecture does give plenty of opportunities for third parties to offer server and client side database encryption as well as improved key management. Examples of such opportunities are discussed in Appendix A.

- Audit and accountability do not appear to be a priority. The SRP authentication algorithm does not offer a non-repudiable proof of login and there is no built in audit trail. Database architects have to add their own auditing using database triggers.

## 2.7.2 Requirements on a Database Management Tool

It is now possible to identify the requirements for security management on a Database Administration tool for day-to-day use by a DBA.

Looking first at the wider picture, the above has identified the following activities that may take place under “Security Management”:

1. Server Deployment including physical security controls; Firebird configuration, choosing the location of security and user databases and ensuring that the filesystem security is suitable for both the location of the Firebird configuration files, plugins and databases, and their use; initialisation and preparation of security databases.
2. Choice of User Authentication schemes and wire encryption.
3. As part of each database's schema definition, the identification of database roles and the assignment of access rights to each such role.
4. Use of server side user database encryption, if any, and the key distribution mechanism for the database encryption keys.
5. User Management including the creation/management of user credentials and the access rights and roles assigned to each user.
6. Monitoring of access rights with particular focus on the identification of “stale” users.
7. Monitoring of User Mappings between authentication schemes.
8. Monitoring, analysis and archiving of Database User Accountability Tables (login/logout logging and other security related events).

Items 1 to 3 are clearly not part of day-to-day DBA activities and are out-of-scope for a day-to-day DBA Management tool.

The same is generally true of item 4. However, key distribution could be considered to be part of User Management. This would be especially true if key distribution using SRP as suggested in Appendix A.1 were implemented. However, the lack of a standard scheme makes it difficult to include this in a general purpose tool at present.

The remaining items are clearly part of day-to-day management activities. However, again the lack of a standard approach for item 8 makes it difficult to include this in a general purpose tool.





# 3

## Using the IBX Services API for Database Administration

In the `ibx/examples/services` directory, an example application is provided that demonstrates the use of IBX for a range of Server Management tasks. These are:

1. Database backup and restore.
2. Viewing the Server Log
3. Viewing Server Statistics
4. User Management
5. Database Validation
6. Limbo Transaction Recovery.
7. Working with Alternative Security Databases in Firebird 3.

The purpose of the example is to demonstrate use of the IBX Services API rather than to provide a practical database administration tool. As regards database administration, using the Services API alone, has not kept pace with the way that Firebird has been developing – for example in the use of virtual tables to access system information. The Database Administration tool presented in section 4 uses virtual database tables wherever possible and the Services API only when it has too. The result is arguably a much “slicker” tool. As may also be observed, using the Services API alone does not meet fully meet the security management requirements for a day-to-day DBA tool identified in 2.7.2.

### 3.1 Running the Example

In the Lazarus IDE, open the project file `ibx/examples/services/services.lpi`. Compile and run.

The program starts by presenting a standard login dialog. By default it proposes the “localhost” as the server, this can be overridden to the DNS name of any server that you have access to.

Note that the example always uses TCP as the connection protocol. To use any other protocol, change the “Protocol” property in the IBServerProperties1 component on the main form before compiling the example program.

You must also enter your login details and password. To use the full range of services available, you will normally want to login as the SYSDBA user. The SQL role “RDB\$ADMIN” is always requested so that if a normal user is logged in they will have Admin privileges enabled for User Management, but only if they have already been granted the Admin Role on the Default Security Database. If you login as a normal user (even with the Admin Role) expect many exceptions to be raised i.e. every time you access a service you are not permitted to use.

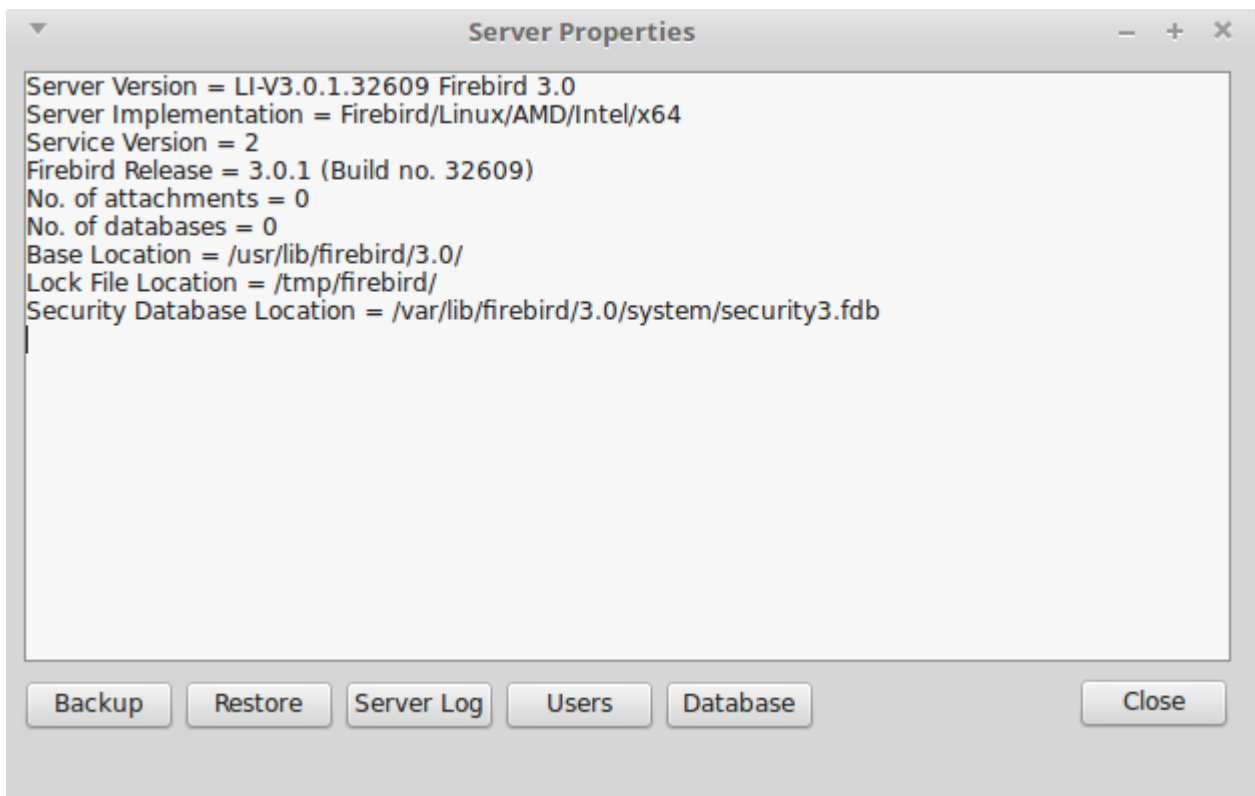


Illustration 1: Services API Example Main Form

On successful login, the IBServerProperties1 component is used to access and display the basic properties of the remote server.

The buttons along the base of the form may now be used to invoke the various actions supported by the Services API.

## 3.2 Database Backup

Selecting the Backup Service brings up the Database Backup dialog.

The Services API component TIBBackupService can be used to backup any database on the server to a *gbak* format archive file. The archive file can be placed on either the server or the local client (your computer).

In order to take a backup:

- enter the database name (the dialog defaults to suggesting the example employee database).

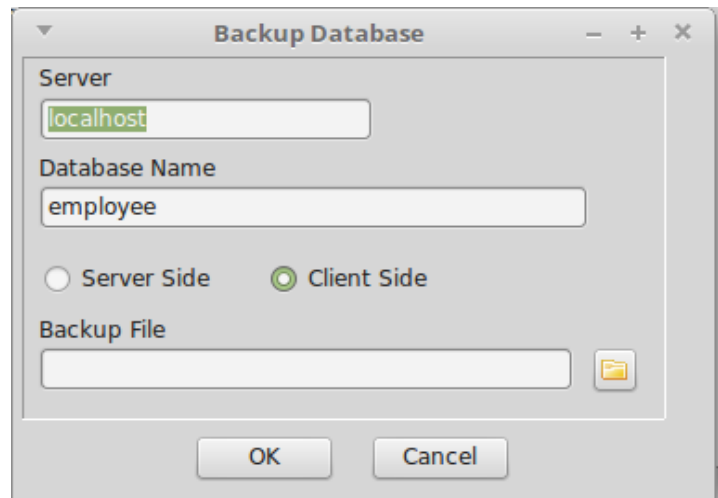
**Note:** that this does is not a connect string and must not include the server name. When using the Services API, the database name is always the name of the database on the selected server.

- Select a Server or Client side backup.
- Enter the backup file name, and
- click on OK.

The backup will now be performed with feedback written to the main form's information log.

Note that with a client side backup, there is no verbose output and the operation completes by recording the number of bytes written.

The action will fail if you login under a user name with insufficient privilege to backup the database.

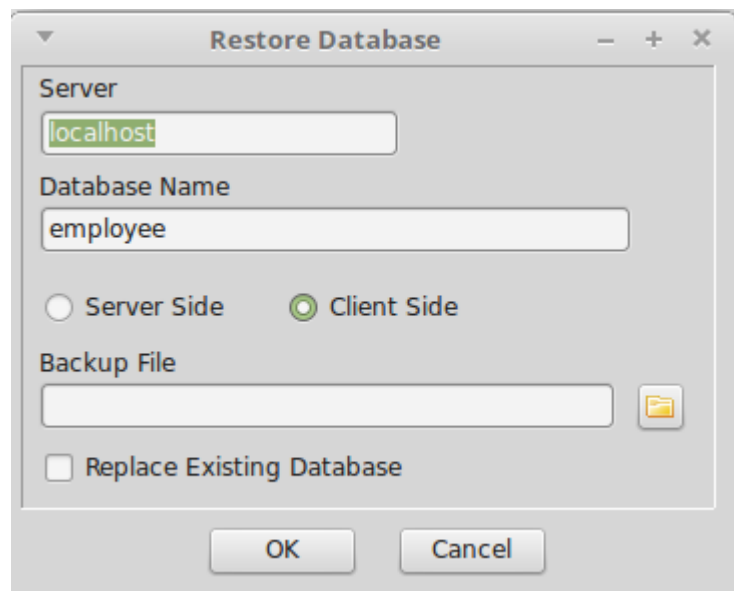


**Illustration 2: Database Backup Dialog**

### 3.3 Database Restore

Selecting the Restore Service brings up the Database Restore Dialog. This uses the TIBRestoreService to create/replace a database from an existing *gbak* format archive and is very similar to the Backup Dialog. The different is that the source archive file must exist. If the destination database exists then the “replace database checkbox must be checked to permit the database to be replaced.

Click on OK to restore the database from the archive. In this case, restore from a client file will generated verbose output.



**Illustration 3: Database Restore Dialog**

### 3.4 Server Log

Click on the “Server Log” button invokes the TIBLogService in order to read the current Log File on the remote server. The results are displayed in the main form's information log.

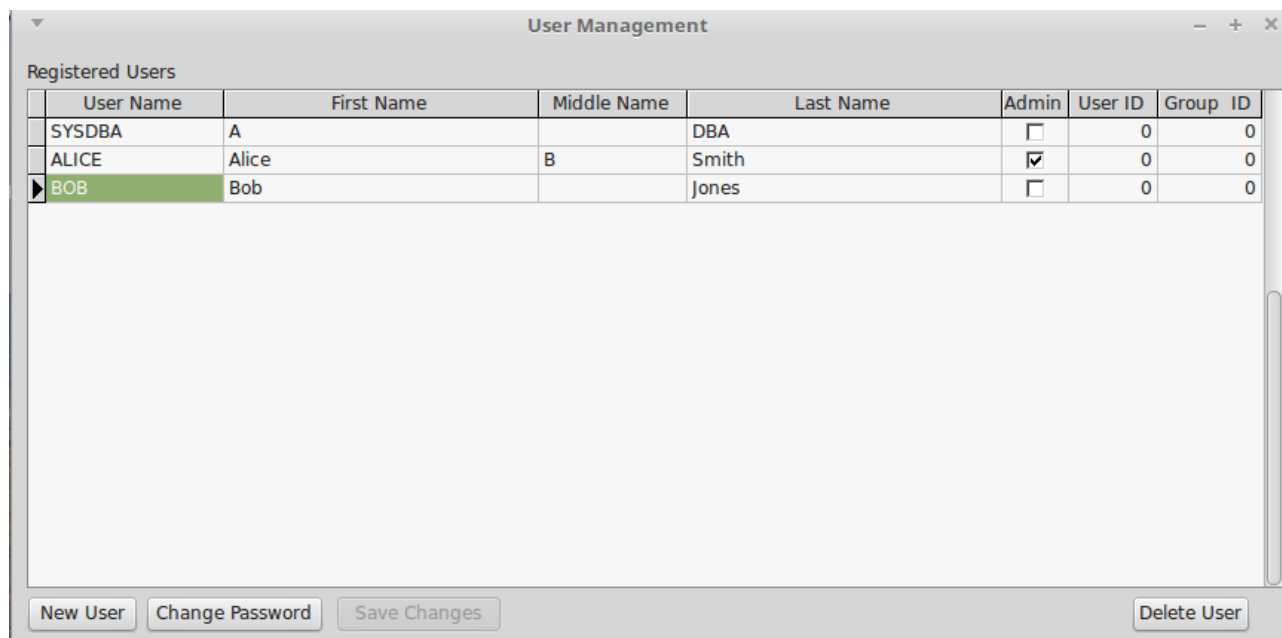
### 3.5 User Management

Clicking on the “Users” button brings up the User Management Dialog.

The TIBSecurityService Component is used to display the current list of users. It may also be used to add a new user, delete and existing user or to change user details.

Prior to Firebird 3, the Services API was the only mechanism available for User Management. In Firebird 3, the SEC\$USERS virtual table has been introduced, which along with the CREATE/ALTER/DROP USER statement enables User Management through the Database connection. Use of the Services API for User Management has been deprecated in Firebird 3 and is anyway limited to the default User Management and the default Security Database.

For an example using the new Firebird 3 features, see section 4.



**Illustration 4: User Management Dialog**

The User Management dialog lists all users managed by the default User Manager in the default Security Database, provided that the logged in user is the SYSDBA or has the Admin Role privilege. Otherwise, the list is limited to the logged in user only.

You can edit any other user information other than the User Name itself. The SYSDBA can also grant the Admin Role to another user. Click on the "Save Changes" button to save edits. Edits are automatically saved when a new row is selected.

The buttons at the base of the dialog allow you to:

- Add a "New User": This opens the Add User Dialog through which the username and password are entered. The remaining user details may be edited as above.
- Change Password: This opens the change password dialog which allows the password for the currently selected user to be changed.
- Delete User: This will delete the currently selected user from the security database.

### 3.6 The "Database" Actions

Clicking on the "Database" button results in a popup menu being displayed from which you can select one of the following actions:

- Show Statistics
- Validation
- Limbo Transaction Resolution
- Database Sweep
- Bring Database Online
- Shutdown Database

### 3.6.1 Show Statistics

This uses the TIBStatisticalService to display the database statistics for a selected database.

Selecting the action results in the “Select Database” dialog being displayed. Enter the name of the database for which statistics are required. The results are displayed in the mainform's information log.

### 3.6.2 Validation

This uses the TIBValidationService to validate (and repair) the database and is similar to the use of the *gfix* utility for validation and repair. This example is limited to running *gfix* with the -v and -f options alone. A more complete example is given in the next section.

Selecting this action brings up the “Select Database” Dialog for validation. This allows the database to be specified and offers a choice between “Full Validation” and “Online Validation”.

Online Validation was introduced in Firebird 3 [2].

Clicking on OK starts the validation task. The results are displayed in the mainform's information log.

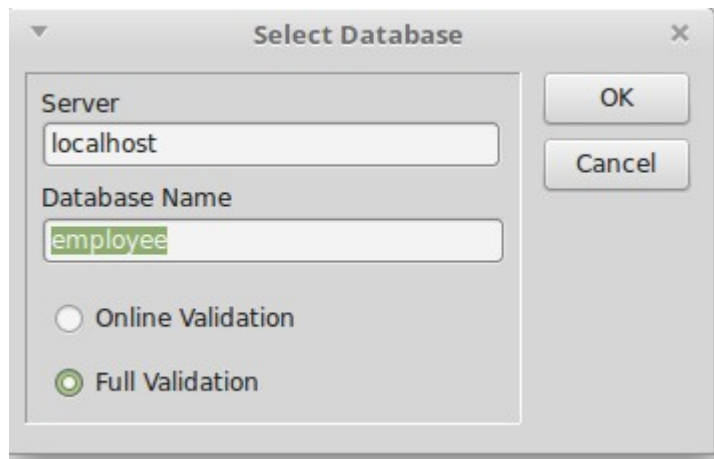


Illustration 5: Database Selection Dialog for Validation

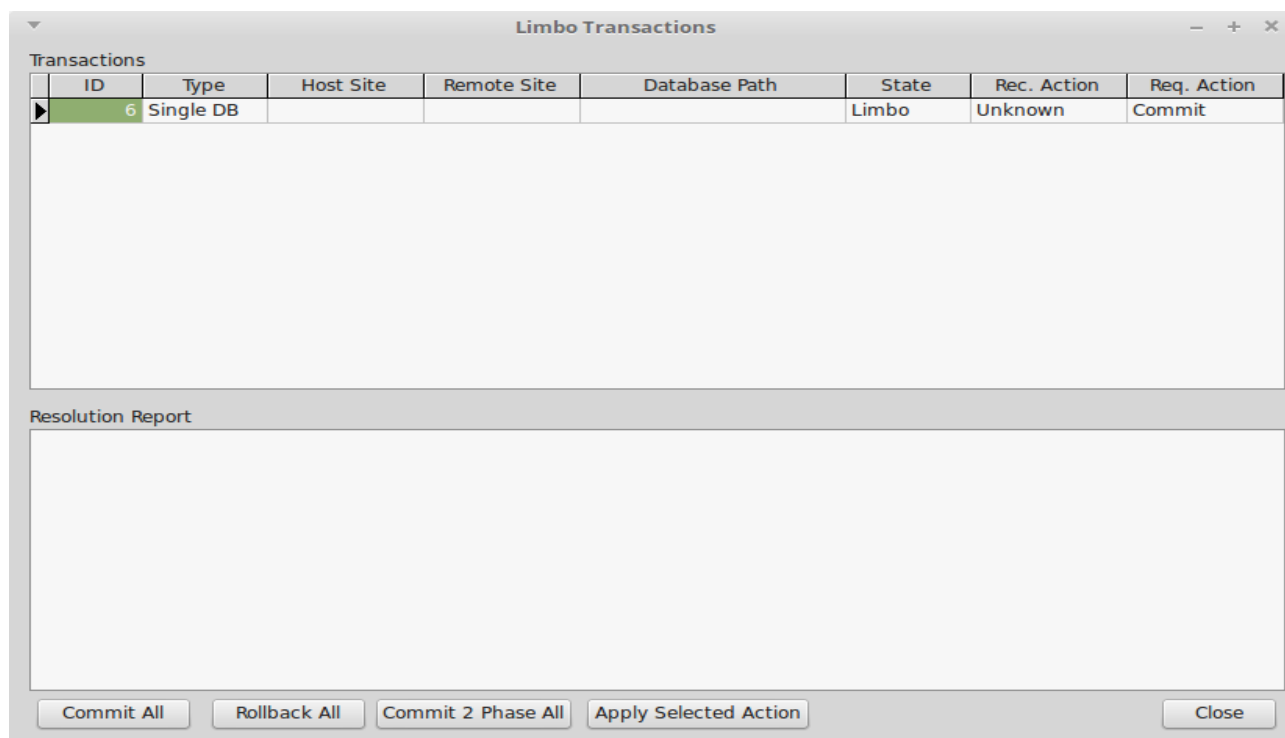
### 3.6.3 Limbo Transaction Resolution

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.

The TIBValidationService may be used to resolve Limbo Transactions.

Selecting this service first opens the “Select Database” dialog and once a database has been selected, Limbo Transactions, if any are identified and displayed using the “Limbo Transactions” dialog.



**Illustration 6: Limbo Transaction Resolution Dialog**

If any are found then they are displayed in the grid shown in Illustration 6 together with a recommended resolution action. This can be overridden by selecting the requested resolution action from the drop down list provided in the “Action” column.

The actual resolution is carried out by selecting one of the resolution actions given at the bottom of the dialog:

- **Commit All:** All Limbo Transactions are committed regardless of the requested/recommend resolution.
- **Rollback All:** All Limbo Transactions are rolled back regardless of the requested/recommend resolution.
- **Commit2 Phase All:** This performs automatic two phase recovery regardless of the requested/recommend resolution. See the *gfx* documentation for further information.
- **Apply Selected Action:** The requested/recommend resolution is performed for each transaction.

The results are displayed in the Resolution Report.

### 3.6.4 Database Sweep

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.

A manual sweep of a database is also possible. The `TIBValidationService` component is used to perform a manually requested Database Sweep.

After selecting the sweep action, the user is prompted to enter the database name of the database on which to perform the sweep. The results are displayed in the mainform's information log.

### 3.6.5 Database Shutdown

When a database has been shutdown it is marked as inaccessible to normal users. This can be useful when preparing for some maintenance activities. The TIBConfigService can be used to perform a database shutdown.

When the action is selected the Database Shutdown dialog is displayed.

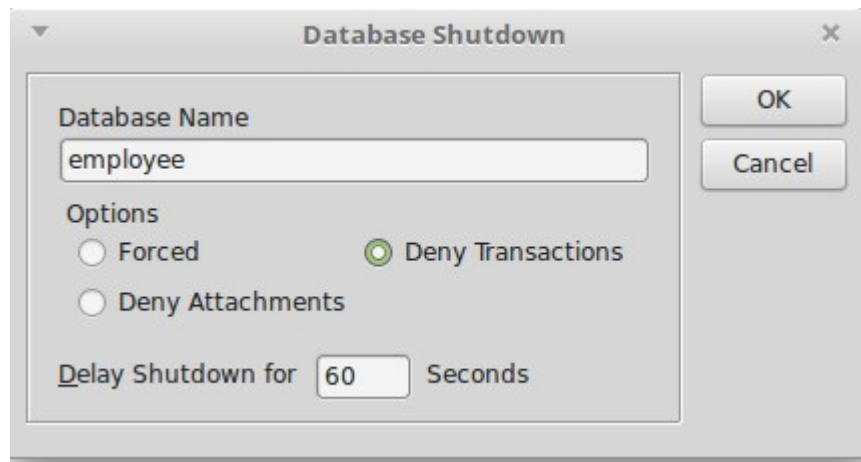


Illustration 7: Database Shutdown Dialog

This allows the user to select the database to be shutdown as well as the Shutdown options and delay time.

The shutdown timer is a delay timer that allows the server to wait for normal users to disconnect from the database before performing the shutdown. If the command cannot complete in the specified time, the shutdown is aborted. The shutdown action is performed either when the last normal user has logged out or when the delay timer expires. It is performed immediately if no normal users are logged in when the shutdown is requested.

The shutdown options are:

- **Forced:** The database is placed offline and any normal users are forcibly disconnected with possible data loss.
- **Deny Attachments:** No new connections are accepted during the delay period.
- **Deny Transactions:** No new connections or transactions are accepted during the delay period.

In the latter two cases, if normal users are still logged in at the end of the delay period then the shutdown fails.

The Services API implements shutdown as a synchronous call and does not return until it has completed. In order to maintain responsiveness, the example application calls the shutdown service in a separate thread.

### 3.6.6 Bringing a Database Backup Online

This is simply the reverse procedure to shutdown and marks the database as being online and available for normal users. The TIBConfigService is also used to perform the service.

When selected, the user is prompted to enter the name of the database to being back online and the service is invoked. It always returns immediately to report success or failure (usually because the database was already online).



### 3.7 Using an Alternative Security Database

Firebird 3 has introduced the ability to configure alternative security databases to control access to one or more groups of databases, separate from those databases for which access is controlled by the default security database (see 2.4.1.4).

The example application initially connects to the server and implicitly uses the default security database to authenticate the login. The user credentials thus entered should be valid for the SYSDBA or a normal user with the Admin Role when using the default security database.

If later on an action is requested (e.g. database backup) for a database that uses an alternative security database, the action will fail with an *isc\_sec\_context* EIBInterBaseError exception.

There are two strategies for handling this:

1. As described in 2.4.4, set up a mapping (in the database using the alternative security database) that maps the logged in user (e.g. the SYSDBA) into the same user with the same privileges in the alternative security database. The example should then work seamlessly with no *isc\_sec\_context* EIBInterBaseError exception.
2. Login to the Services API with the *isc\_spd\_expected\_db* login parameter specifying the database using the alternative security database. In this case, the login user and password must match the SYSDBA (or a user with the Admin Role) in the alternative security database. Once logged in, the Services API may then be used for operations on the database without incurring an *isc\_sec\_context* EIBInterBaseError exception.

The former strategy may be demonstrated using the example application, but has to be set up by creating the necessary mapping using an appropriate SQL statement and executed using (e.g.) the Firebird *isql* utility. However, the example application does demonstrate the second strategy.

The algorithm used is:

- An *isc\_sec\_context* EIBInterBaseError exception is raised when an attempt is made to perform an action which requires login to the alternative security database.
- The exception is trapped. The current Services API connection is dropped and a new login attempt made. The user is prompted with a revised login dialog indicating the reason for the login and a request to enter valid user credentials for the alternative security database.
- When the actual login is performed, the “expected\_db” parameter is added to the service login parameters and with the database name as its value.
- Once the login succeeds, the original action is attempted again.

The purpose of the above is to demonstrate a strategy for managing the problem. Other, perhaps more elegant algorithms may be possible. Note that if a service is later requested to a different database using a different security database then the above is repeated.










### 3.8 How it works

This is intended to be a simple example of the use of the Services API. However, the packaging of the API into separate services components can obscure some of the simplicity.

The IBX IBServices unit and its components (e.g. TIBServerProperties) date from the original IBX shipped with Delphi and were introduced in order to support the Services API which, itself, was introduced with InterBase 6. The Services API uses a separate connection to that used to access a database and is subject to a separate login exchange. Once the connection is opened, the services protocol is used to:

- Start a service. Depending on the service this may give a single response or a series of responses.
- Check the status of a running service
- Request a response from a running service.

Instead of providing a single component able to invoke any service and receive its responses, the design delivered a set of components, each supporting a sub-range of the services available:

	TIBBackupService	The backup service supports database backup to gbak format archives. Both server side and client side backup file locations are supported.
	TIBRestoreService	The restore service supports database restore from gbak format archives. Both server side and client side backup file locations are supported.
	TIBConfigService	The configuration service allows database parameters to be modified, including whether the database is online, sync versus async writes, etc.
	TIBServerProperties	This service retrieves various server properties including the server version information, server parameters and the current status of database attachments.
	TIBLogService	This service supports the retrieval of the server log file contents.
	TIBStatisticalService	This service supports the retrieval of per database statistics.
	TIBSecurityService	This service supports management of the User Security Database.
	TIBValidationService	This service supports the invocation of various database repair actions, including validation and sweep. Limbo Transactions can also be resolved.
	TIBOnlineValidationService	This service was introduced for Firebird 3 and provides for a table level validation of a database. It implements consistency checks that do not require exclusive access to a database.

### 3.8.1 Services API Login

While this approach gives a neat subdivision between the services available, it hides the fact that they can all use a common connection. Indeed, the original design forced each component to create (and login to) its own Services API connection. There was no mechanism for sharing the same Services API connection.

This meant that either the program designer had to copy all the login parameters between each component or force a separate user login for each component in use.

**Note:** an IBServices component logs into the service when its Active property is set to true.

### 3.8.2 IBX for Lazarus and the Services API

IBX for Lazarus 1.x kept the original IBX model for the Services API. However, IBX 2 repackaged all low level functions, including use of the Services API into the *fbintf* package with separate variants for the new Firebird 3 client library API and the legacy Firebird 2.x version implemented using COM reference counted interfaces. The IBServices components became little more than wrappers around the low level functions.

Each component now had a reference to the low level *fbintf* interface (the *ServiceIntf* property) and this interface could be copied from one IBServices component to another allowing them to share the same Services API connection and hence simplify the program using them, whilst still retaining backwards compatibility.

However, the implementation still had two limitations: One was that that assigning the *ServiceIntf* did not update any other properties which could now be inconsistent (although they would normally be ignored). The other was that the original behaviour on setting the component's Active property to false was retained. That is that the Services API connection was dropped – for every component using it. That meant that care had to be taken as to when to set this property.

### 3.8.3 IBX 2.2 and the Services API

IBX 2.2 has introduced an “Assign” method to each of the IBServices components (inherited from *TPersistent*). When this is used to assign one IBServices component to another it copies the *ServiceIntf* and all the connection related properties from the source to the destination service thus ensuring consistency.

IBX 2.2 has also changed the effect of setting the component's Active property to false. All this does now is to set the *ServiceIntf* to nil. This changes the disconnect from a “first one out” to a “last one out” strategy. That is when the *ServiceIntf* is shared, instead of being dropped the first time a using component's Active property is set to false, it is only dropped when all using component's have their Active property set to false. This is due to the interface being reference counted.

The example program makes full use of the Assign method to share the Services API connection between the components.

### 3.8.4 User Management

The *TIBSecurityService* is not a *TDataset* descendent and cannot be used as the source for an editable *TDBGrid*. In order to create an editable list of users, a *TMemDataset* is used as an intermediary.

When the dataset is opened, it uses the `TIBSecurityService` to get a list users from the server. These are then added to the dataset, one per row. A `TMemDataset` is editable and this allows for in-situ editing of user attributes, or even the deletion of an entire row.

Any such changes are applied to the security database in the `BeforePost` or `BeforeDelete` events for the `TMemDataset`. In each case, the `TIBSecurityService` is used to apply the requested change.

The list of roles is sourced by a `TIBQuery` and generated from the `RDB$ROLES` virtual table joined to the `RDB$USER_PRIVILEGES` table in order to identify which roles have been assigned to the user. It is also in a master/detail relationship with the `TMemDataset` used to list the users which allows it to automatically focus on the current user. Roles cannot be added or removed. However, it is possible to grant/revoke a role from the currently selected user.

A `TIBUpdate` component is used to apply changes to the assigned roles. Its “apply” event handler reviews the requested change and formats and executes a `GRANT/REVOKE ROLE SQL` Statement as appropriate.

### 3.8.5 Limbo Transactions

A similar approach using a `TMemDataset` is used for Limbo Transaction resolution. In this case, the source for the dataset rows is a `TIBValidationService` component and its `LimboTransactionInfo` property. Changes made to the dataset are used to update the `LimboTransactionInfo`.

The `TIBValidationService` is used to apply the required resolution.

### 3.8.6 Using Alternative Security Databases

The only change introduced into IBX 2.2 in order to support alternative security databases is to support the “expected\_db” parameter on login. This directs the login to use the security database configured for the database given as the parameter value.

This feature is used in the example program by running all service requests via a “wrapper” method

```
TRunServiceProc = procedure of object;  
function TMainForm.RunService(aService: TIBCustomService;  
                               RunProc: TRunServiceProc): boolean;
```

The semantic is to run the specified method (`RunProc`) and to prepare the specified service for use with the method.

The wrapper method assumes that the `IBServerProperties` component is the “keeper of the Services API connection” and was activated when the program started. It implements the following algorithm:

1. If the requested service uses a database then its `DatabaseName` property is set to the currently requested database.
2. The service is assigned the properties and service interface from the `IBServerProperties` component.
3. “RunProc” is now run.

4. If an *isc\_sec\_context* `EIBInterBaseError` exception is raised, the `IBServerProperties.Active` is set to false and a new login attempt is made using the alternative login dialog and with the `expected_db` parameter set to the requested database. Step 2 is repeated.

### 3.9 Summary

This example is intended to demonstrate use of the Services API and to demonstrate a strategy for handling alternative security databases. However, in Firebird 3, use of the Services API for User Management is deprecated and cannot anyway be used for alternative security databases. The following example is intended to demonstrate database management in a more Firebird 3 compliant manner.



# 4

## The DBAdmin Tool

The purpose of this example is to both demonstrate Database Management using IBX and to provide a usable tool for Database Administration. It may be found in the `ibx/examples/DBAdmin` directory.

Unlike the Services API example, the program is focused on a specific database. While it also uses the Services API for server level functions they are always used in the context of the current database. The program is intended to demonstrate:

- Access to Database Parameters and their management
- Database Backup and Restore
- Database File Management including Secondary Files and Shadow File sets
- Inspection and Management of Database Attachments
- Access to Database Statistics
- Database schema listing
- Access to Server Properties and the Server Log
- User Management
- Monitoring of User Mappings
- Monitoring of User/Role Access Rights including the identification of “stale” user rights.
- Database Validation and Repair
- Limbo Transaction Resolution.

The example is intended to meet the requirements for day-to-day security management identified in 2.7.2. However, this is with the exception of the monitoring of user access logs. The lack of a standard approach makes it difficult to show this in a general purpose tool.

## 4.1 Running the Example

In the Lazarus IDE, open the project file `ibx/examples/DBAdmin/DBAdmin.lpi`. Compile and run.

The program starts by presenting a standard login dialog. By default it proposes to login to the Firebird example employee database on the local server. This can be overridden to give a connect string for any database to which you have access to.

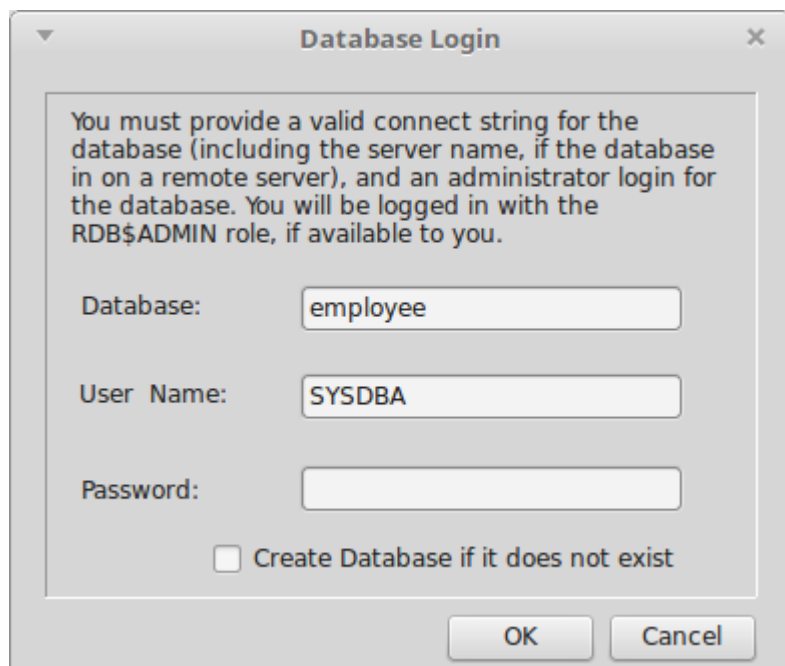
**Note:** in contrast to the previous example which was concerned with Server Login, the full connect string must be provided if the database is on a remote server. For those in doubt the connect string formats supported are described in the Firebird 3 release notes [2].

A valid user name and password must also be provided. This may be the SYSDBA or could be any normal user that has been granted the RDB\$ADMIN role (see 2.5.4) for the database. This role is always requested by this program.

A normal user may still log in but will find many of the functions inaccessible.

The database does not need to exist. If the "Create Database..." checkbox is selected then an empty database is automatically created after a successful login (provided the user is permitted to create a database).

A so created empty database can then be populated by restoring a *gbak* format archive.



**Illustration 8: Database Login Dialog**

Following a successful login, the example program's main form is displayed (see Illustration 9). This uses a TPageControl to structure the presentation into multiple pages.

## 4.2 Database Properties

The initial display shows the database properties. These are sourced from:

- Database Information provided over the Database API connection (using the TIBDatabaseInfo component), and
- The MON\$DATABASE and SEC\$GLOBAL\_AUTH\_MAPPING virtual tables.
- The TIBStatisticalService; used to read the database header information e.g. for the shadow file status.



If the server is Firebird 2.5 or earlier, some of the information is not present (e.g. Pages Used) and replaced with “n/a”.

The screenshot shows the 'Database Administrator Demo' window with the 'Properties' tab selected. The window displays various configuration settings for a database named 'employee'.

Property	Value
Server	zeus
Database Name	/usr/share/doc/firebird3.0-common-doc/ex
Connect String	employee
ODS Version	12.0
Svr Version No.	LI-V6.3.2.32703 Firebird 3.0
SQL Dialect	3
DB Owner	SYSDBA
Sec. Database	Default
Character Set	NONE
Pages Allocated	346
Page Size	8192 bytes
Pages Used	32
Pages Available	314
Current Memory	3038496 bytes
Max Memory	3112496 bytes
No. of Buffers	128
Date DB Created	11-5-17
Linger Delay	0 seconds
Sweep Interval set to	10000 Transactions
Read Only	<input type="checkbox"/>
Online	<input checked="" type="checkbox"/>
Synchronous Disk Writes	<input checked="" type="checkbox"/>
Shadow Database	<input type="checkbox"/>
Space Reserved for Backup Records	<input checked="" type="checkbox"/>
Auto Admin Mapping	<input type="checkbox"/>

Database: employee - Logged in as user SYSDBA by Srp, using Default security database

**Illustration 9: Database Administration Example Application**

Most of the information presented cannot be modified, the exceptions being:

- SQL Dialect
- Character Set (Firebird 3 and later)
- Linger Delay (Firebird 3 and later)
- Sweep Interval
- No. of Buffers
- All checkboxes (Auto Admin is available Firebird 2.5 and later)

In each case, the property may be directly edited and the underlying database property is updated immediately after the change has been made.

- A database marked as a shadow database can be made into a normal database (Activate Shadow function) but not vice-versa. That is the shadow database flag can be unchecked if checked, but not the other way round.
- Unchecking the “Online” checkbox is the same as requesting a Database Shutdown (see 3.6.5) and uses the TIBConfigService. The reverse action brings the database back online.

- Auto Admin Mapping is discussed in 2.5.4.2.

### 4.2.1 Database Backup

The database backup function is an improved version of that provided with the previous example. It is invoked from the toolbar or the Files menu. The function is used to backup the current database only and cannot be used to backup another database.

Both client and server side backups are supported along with many *gbak* options including the “No Database Triggers” option introduced with Firebird 3.

The user must specify backup file.

Click on OK to start the backup. The dialog changes to show the backup log.

On completion, the dialog may be closed.

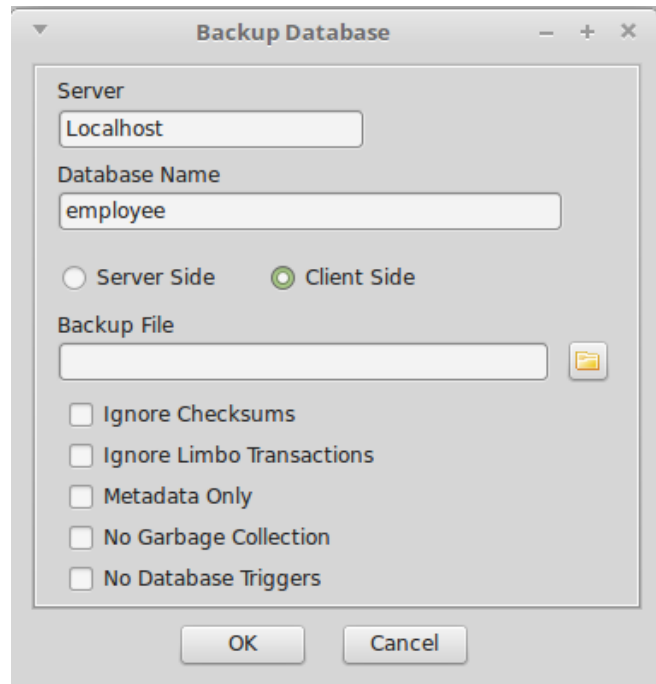


Illustration 10: Database Backup Dialog

### 4.2.2 Database Restore

The database restore function is an improved version of that provided with the previous example. It is invoked from the toolbar or the Files menu. The function is used to replace the current database only and cannot be used to create a new database.

**Note:** a new empty database can be created when logging into a non-existent database.

The user must specify the backup file from which the restore takes place. This file must already exist.

The user is presented with the current database Page Size and the number of Page Buffers. These can be modified when a database is restored. Beware: inappropriate settings can adversely affect performance and disk usage.

Both client and server side archive files are supported along with many *gbak* options.

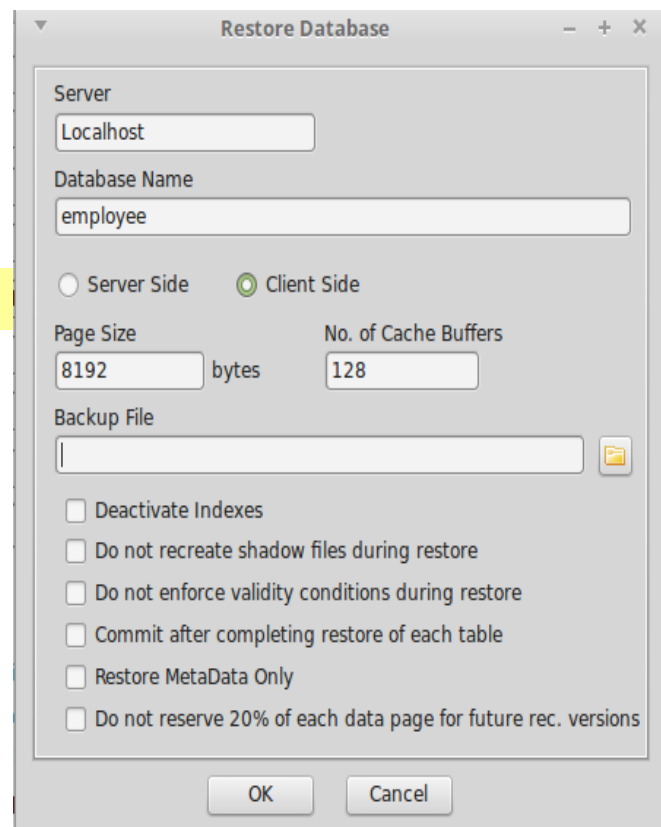


Illustration 11: Database Restore Dialog

## 4.3 The Files Page

The Files Page is used to display information about which files are used for the database and also allows secondary files to be added and the management of Shadow File sets. Metadata tables are used to source the secondary file and shadow file set information.

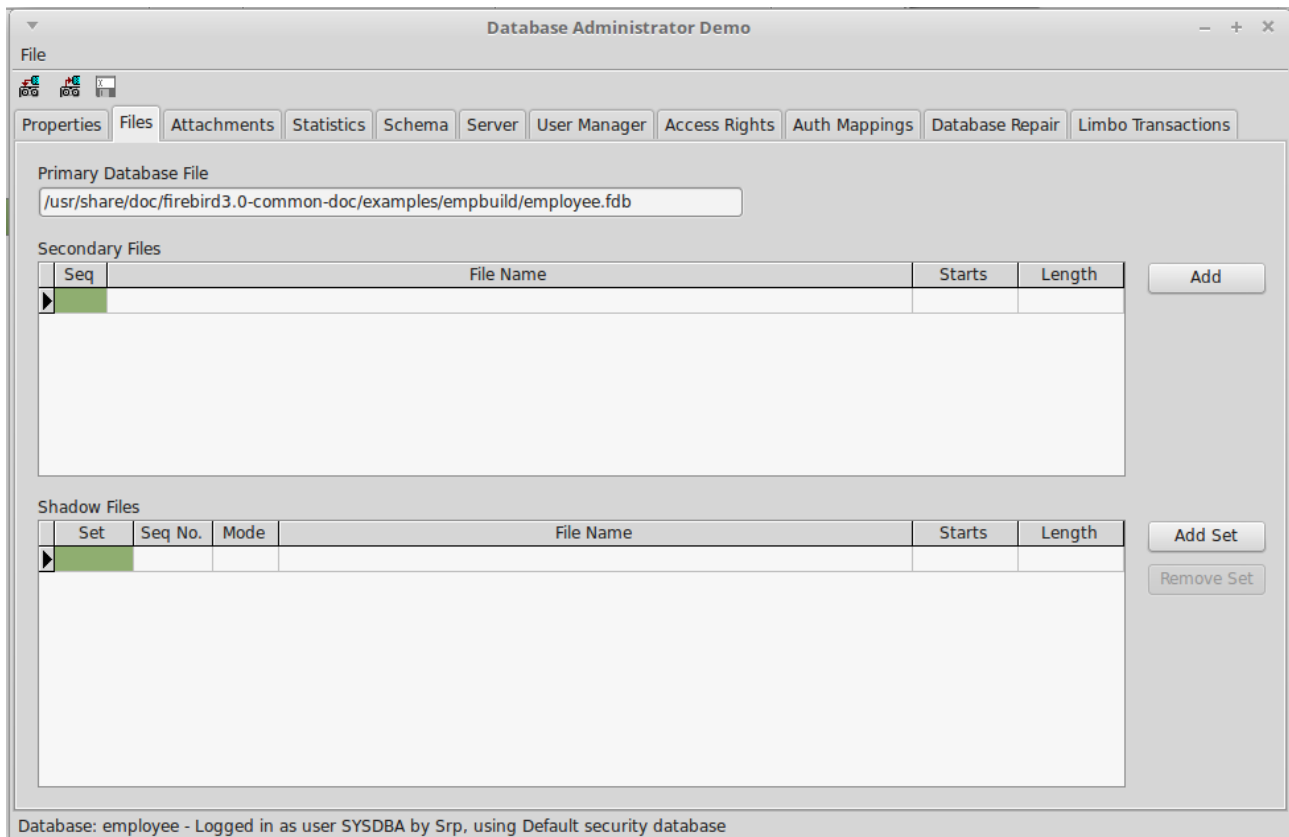


Illustration 12: The DBAdmin File Page

### 4.3.1 Adding a Secondary File

If the database file is running out of usable space on its current filesystem then it is possible to add secondary files on other filesystems so that the database can grow beyond the limits of a single filesystem. This operation can only be performed when the database has been shutdown and is exclusively available to the Database Administrator.

To add a secondary file, click on the “Add” button. This brings up the Add Secondary File dialog.

The filename must be a valid pathname on the server and at a location that the server administrator has permitted database files to reside (see 2.2).

The “Starting After” should be set with reference to the number of pages already allocated to the database (see Database Properties) dialog. If you choose to specify the starting point in MBs then the starting page number will be calculated for you.

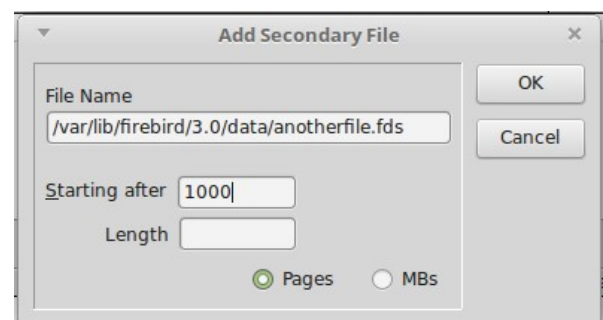


Illustration 13: Add Secondary File Dialog

The “Length” should be left empty if this is the last file added and is always optional.

The SQL statement “Alter Database Add File...” is used to add the secondary file. The documentation of this statement [8] should be consulted for more information.

### 4.3.2 Shadow Sets

A shadow is an exact, page-by-page copy of a database. Once a shadow is created, all changes made in the database are immediately reflected in the shadow. If the primary database file becomes unavailable for some reason, the Firebird Server can automatically switch to the shadow.

Like a database, a shadow may be multi-file. The number and size of a shadow's files are not related to the number and size of the files of database it is shadowing.

If an unrecoverable error occurs on the primary database file(s) then the server will convert a shadow set to being the primary database. The subsequent action depends on the shadow mode specified for the Shadow Set i.e.

- **AUTO** (the default value): shadowing ceases automatically as soon as a Shadow set becomes unavailable either because it has been converted into the primary database or for any other reason; all references to it are deleted from the database header and the database continues functioning normally.
- **CONDITIONAL**: when the shadow becomes unavailable, the system will attempt to create a new shadow to replace the lost one. If it does not succeed, a new shadow may need to be created manually.
- **MANUAL**: when the shadow becomes unavailable, all attempts to connect to the database and to query it will be rejected with an error message. The database will remain inaccessible until either the shadow again becomes available or the database administrator deletes it using the DROP SHADOW DDL statement.

Note: MANUAL should be selected if continuous shadowing is more important than uninterrupted operation of the database.

Manual mode creates a problem for the DBAdmin Tool as it is prevented from logging into the database if a MANUAL mode shadow becomes unavailable and is hence prevented from taking the necessary recovery action.

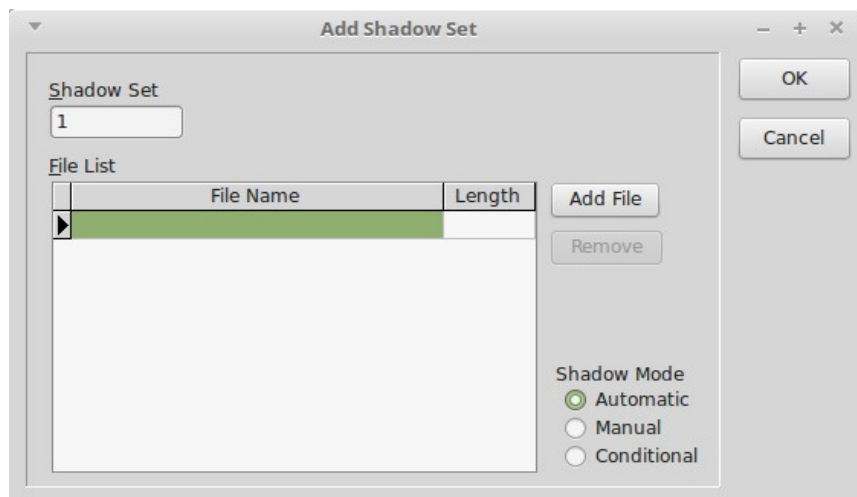
Unavailability of a shadow is reported at database login time as an I/O exception plus a (hopefully) descriptive message. However, this is not the only reason for an I/O exception at login. When such an exception occurs, the user is presented with the error message and asked if they want to “kill” all unavailable shadows (equivalent to running *gfix* with the -kill option). If they answer “yes” then the Services API is used to kill all unavailable shadows (on the database) and a new login attempt is made. If the user answers “no” then control returns to the login dialog.

Note: The kill unavailable shadows command is also available on the Database Repair tab.

#### 4.3.2.1 Adding a Shadow Set

A new Shadow Set may be created at any time. The database does not have to be taken offline and normal activity may continue while the shadow set is being prepared.

Clicking on “Add Set” displays the Add Shadow Set Dialog (see Illustration 14). You need to enter the full path for one or more files in the server. These files must reside in a location that the server administrator has configured as available for use by Firebird. A Shadow Set may comprise a single file or a primary file plus secondary files. In the latter case then length of each file other than the last must be given.



**Illustration 14: Add Shadow Set Dialog**

The Shadow Mode should be selected (see above). And then click on OK to create the shadow set.

This may take some time depending on the size of the database.

#### 4.3.2.2 Dropping a Shadow Set

Select the Shadow Set to be dropped and click on remove Shadow Set. As with adding a shadow set, this can be done at any time. You will be prompted whether or not to delete the shadow files as well (Firebird 3 and later).

## 4.4 The Attachments Page

This page allows the DBA to monitor connections to the database (See Illustration 15).

A TIBDynamicGrid is used to display the list of attachments, which is sourced from the MON\$ATTACHMENTS virtual table. It can be sorted by any column.

In addition, any row can be expanded to reveal information about the connection that cannot be accommodated on the main row.

A right click menu option allows a connection (other than the one used by the DBAdmin tool) to be deleted. This is performed by a DML Delete statement acting on the current row of the table.

Another option on the right click menu allows the DBA to request that this tab is automatically refreshed every five seconds in order to monitor the current set of attachments in near real time.

## 4.5 The Statistics Page

The statistics page allows the display of database statistics selected from a drop down list of options.

- Header Only: Information sourced from the Database Header Page
- Performance and Operational: statistics sourced from the Database Information API
- Header and Data: Database Header plus data pages (tables)
- Header, Data and Indexes: Above plus indexes.
- All: As above but includes System tables.

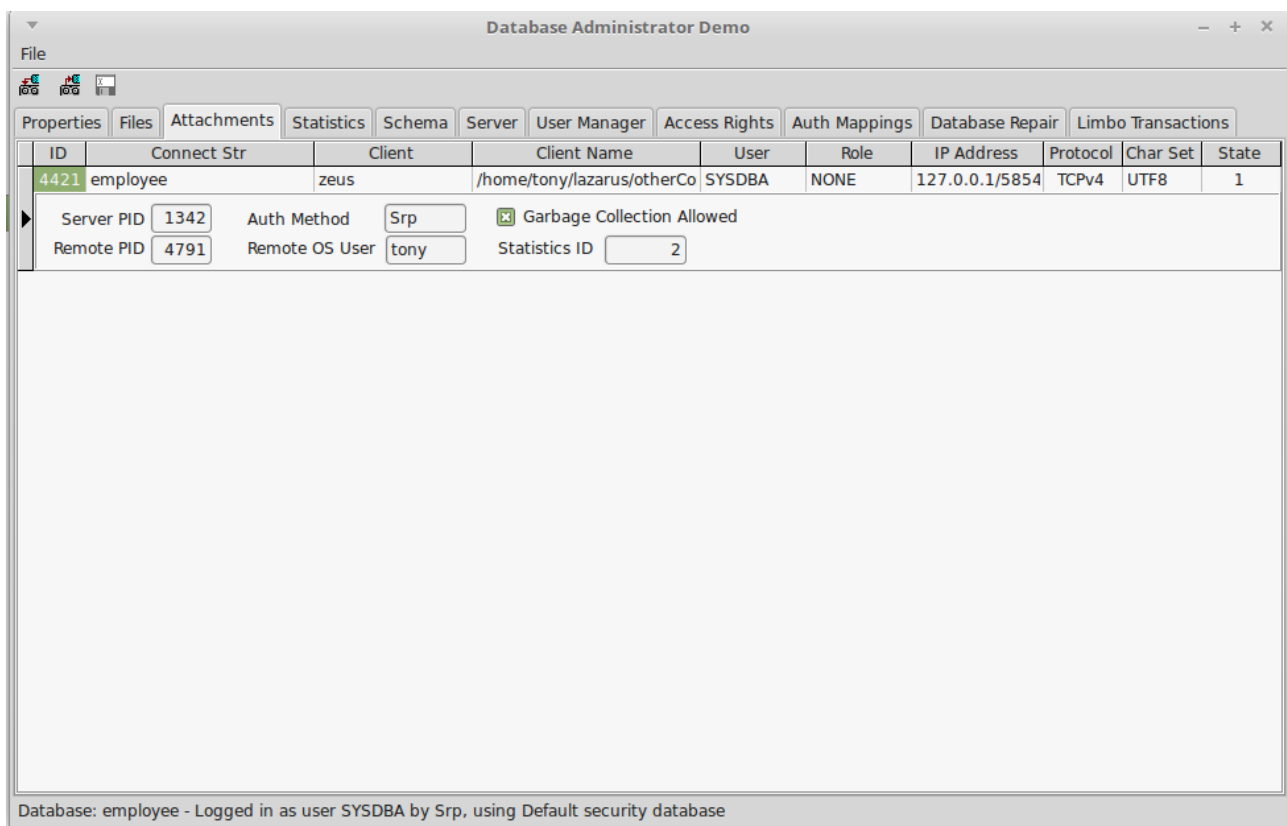


Illustration 15: Database Attachments Page

## 4.6 Schema Page

This page uses TIBExtract to dump the current database metadata to a SynEdit read only view with syntax highlighting. The page contents can be saved to a file.

The schema is by default extracted without grants to users other than “PUBLIC”. This is because these are not considered part of the schema itself; a database with the same metadata in a different organisation would have a different set of users. However, it is possible to include these grants by checking the “Include Grants to Users” checkbox at the bottom left of the page.

## 4.7 The Server Page

The Server Tab is used to display the server properties and the server log. As with the Service API example, this uses the TIBServerProperties and the TIBLogService to access the information.

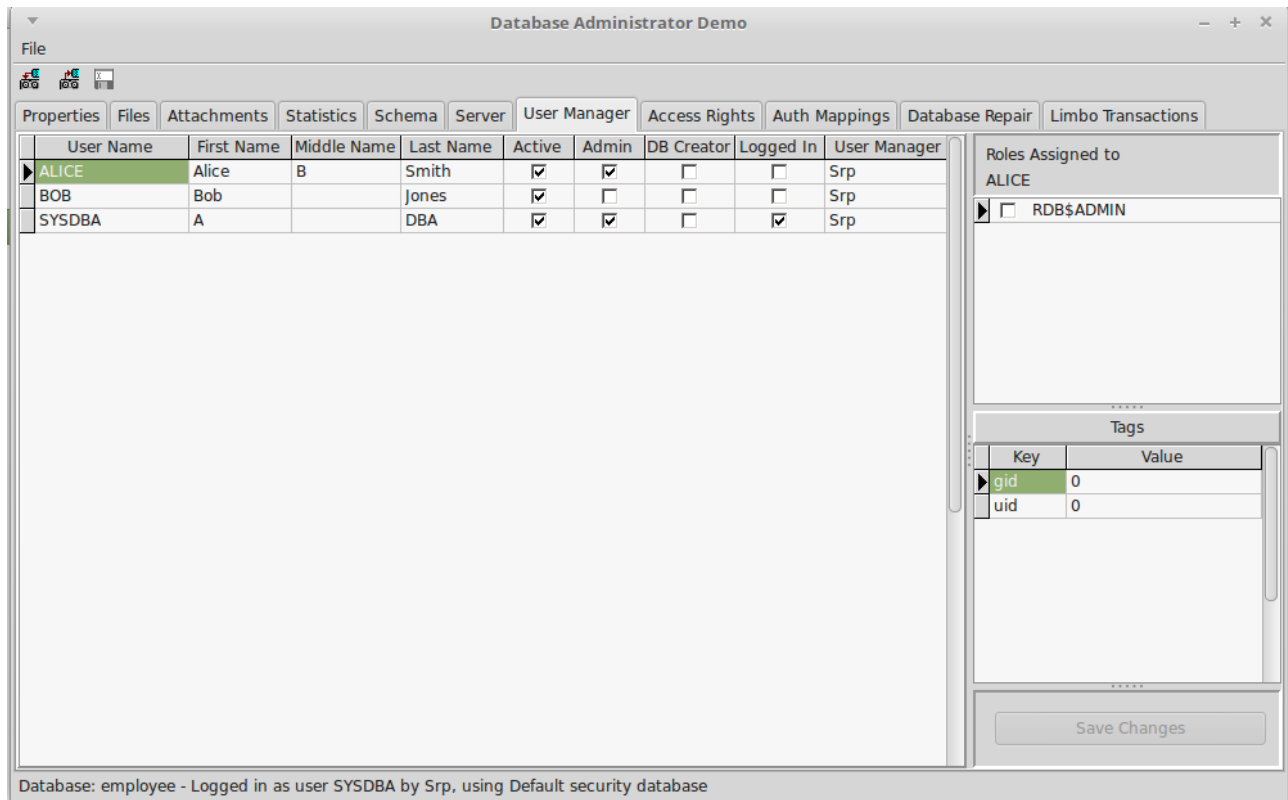
## 4.8 The User Manager Page

The User Manager Page is designed to work with either the Services API or the SEC\$USERS Table. The former is used for Firebird 2.x servers or when the current user has the Admin Role configured in the security database but has not been granted the RDB\$ADMIN role for the current database.

Note that when an Alternative Security Database has been configured for the current database, the list of users in the SEC\$USERS table comes from this database and not the default security database.

At design time, TIBDynamicGrid used to display the user list has a superset of columns defined for each of the different modes. When the database is opened, the server version is checked and the

appropriate set of columns made visible. The data source is directed either to a TMemDataset, or a TIBQuery (for the SEC\$USERS table) as appropriate.



**Illustration 16: The User Management Tab**

Entries may be edited in-situ and the right click menu may be used to add or delete users, or to change a user password.

The following attributes are given by a checkbox:

- Active: Only “active” users are permitted to login to a database.
- Admin: User has the Admin Role granted to the security database from which the user list is sourced. (SYSDBA is automatically the administrator regardless of this setting).
- DB Creator: the user has the “Create Database” privilege granted to them.
- Logged in: the user is logged in at least once to the database.

When the source is the TMemDataSet, updates are applied as described in 3.8.4. When the source is the TIBQuery then a TIBUpdate component is used to Add, Edit or Delete a user. This generates and executes the appropriate CREATE/ALTER/DROP USER statements.

Note that when a new user is added, the User Manager Plugin (Firebird 3 only) can be selected from a drop down list of plugins. This is a hard coded list given that there is no easy way to determine the list of available User Manager Plugins.

The handling of assigned roles is also as described in 3.8.4.

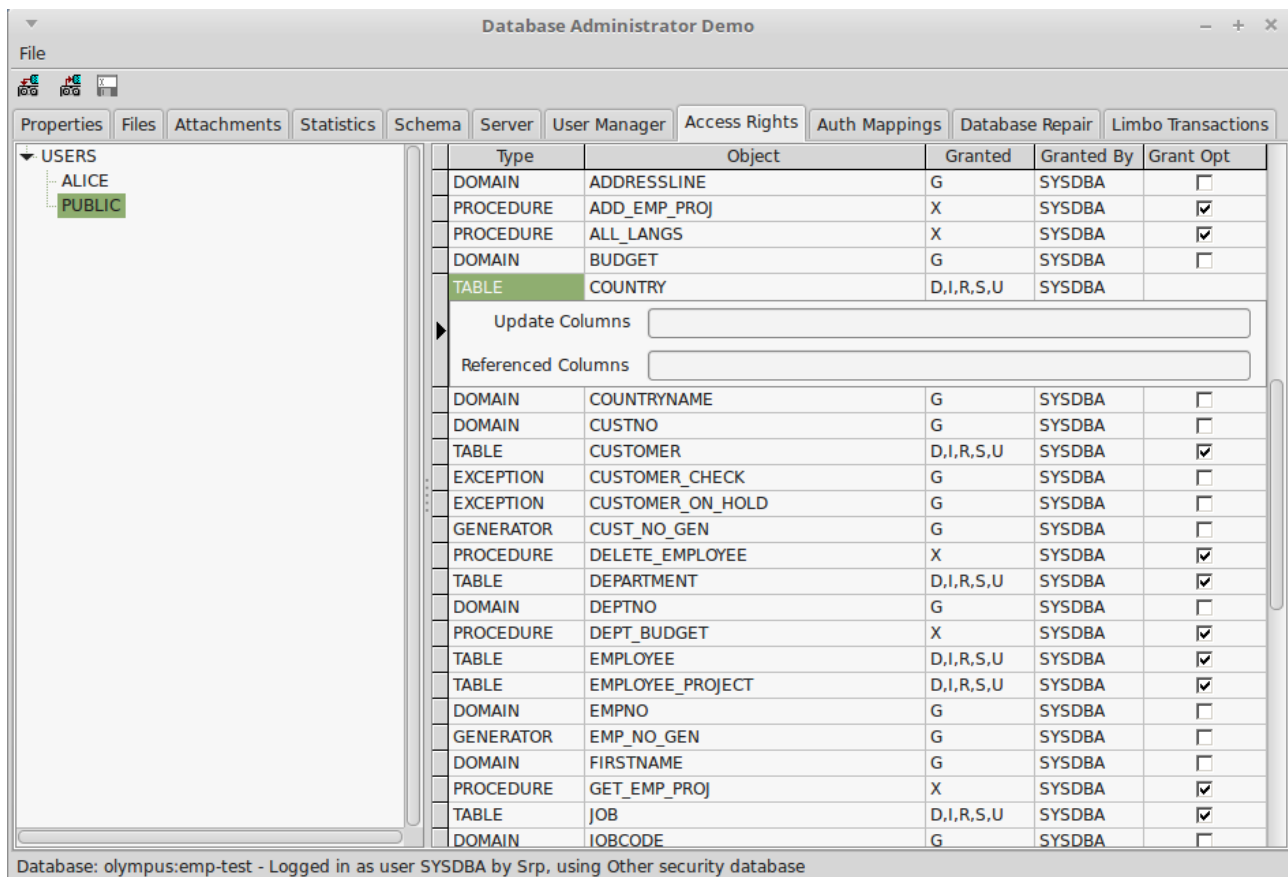


Before deleting a user, you should remove all roles from that user in each database in which they have been granted a role. Otherwise, the user will appear as a stale user in the Access Rights Page (see below).

## 4.9 The Access Rights Page

The purpose of the Access Rights Page is to allow the DBA to:

- Monitor the current access rights assignment in the database, and
- Identify and remove any “stale” user access rights.



**Illustration 17: Access Rights Page**

The tab has two window panes. The left hand pane is used to display “subjects” to whom access rights have been granted together with their dependent objects in a hierarchical display. The hierarchy reflects the fact that some objects can be both subjects and objects (e.g. Roles and Stored Procedures). There is also a top level grouping into object categories such as “Users”, “Roles”, “Triggers”, etc.

A top level category only appears when access rights have been granted to an object in that category.

The right hand pane displays the detailed access rights granted to the object selected in the left hand pane. This lists, for each object to which access rights have been granted:

- The Object Type (e.g. Relation (Table), Procedure, Role, Exception, etc.)

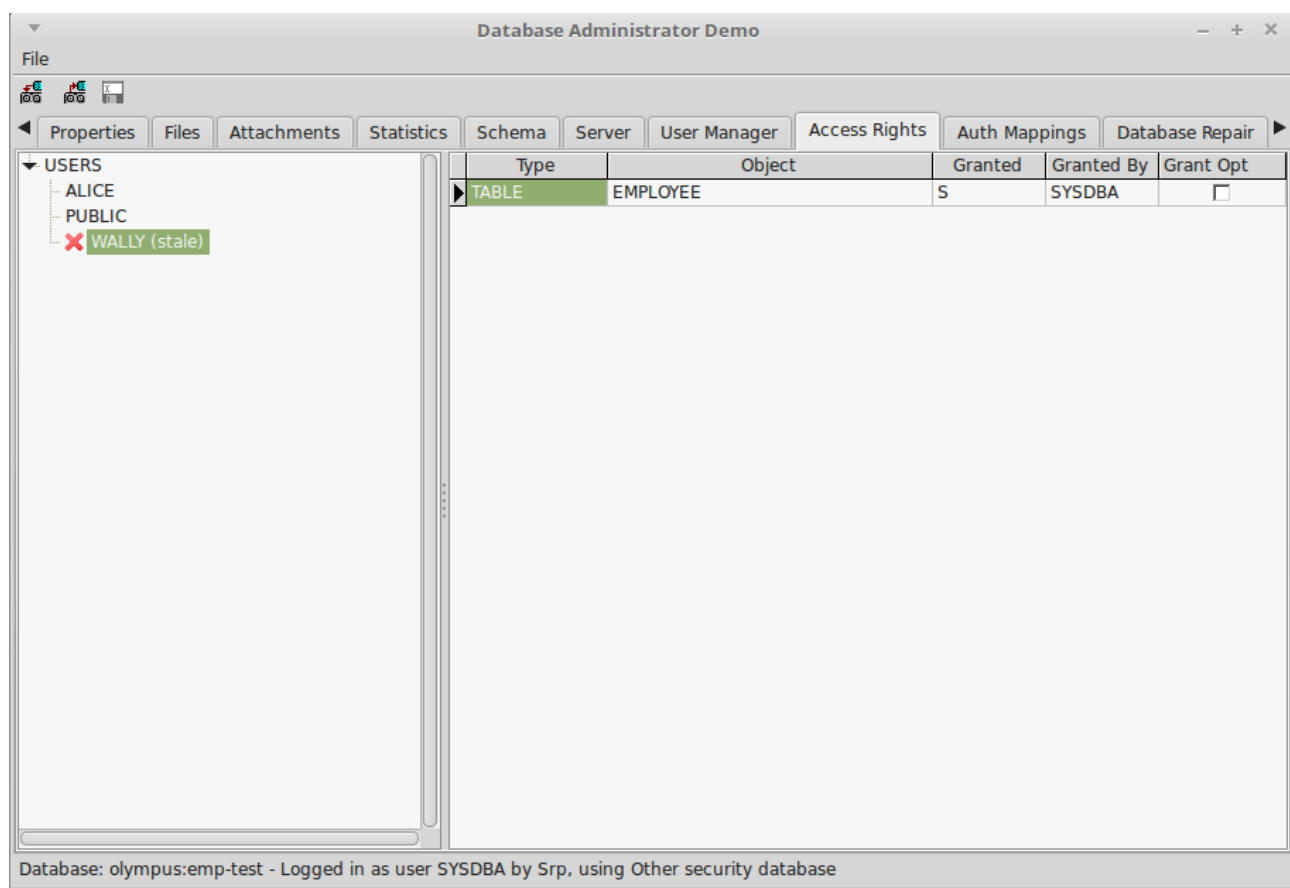


- The Object Name
- Access Rights granted coded by their letter identifier and given as a list.
- Who granted the access right, and
- Whether the grant option was also given.

An editor panel is used to display the list of update columns and/or referenced columns, if specified, below the current row. The Update and References privileges may be granted for a restricted set of columns or to all columns. In the former case, the list of columns to which the Update/References privilege is granted appears on the editor panel. If the Update/References privilege is not granted, is not appropriate for the object, or applies to all columns then the list is empty.

The editor panel appears automatically when a row is selected with a non-empty list of update/references columns. It can also be revealed by clicking on a row's indicator column (to the left of the row).

#### 4.9.1 Stale Users



**Illustration 18: State Access Rights Indication**

The risk of “stale” user access rights existing in a database was identified in 2.6.3. This is when access rights are granted to a user that does not exist in the security database. It is preferable that these are removed as soon as possible as they risk a new user being created with the same name who then inherits inappropriate access rights for their role.

Stale users are indicated in the left hand pane by the text “(stale)” appearing after their name and a red diagonal cross symbol to their left. (see Illustration 18 where the user “Wally” cannot be found in the list of users in the security database). When a stale user is selected, the right hand pane lists the access rights they have been granted.

The DBA Tool can remove access rights from these (or any other) user. A “Revoke All” function may be selected from the right click popup menu, in the left hand pane, and when invoked creates and executes “Revoke” statements to remove all access rights granted to that user.

## 4.10 The Auth Mappings Page

The purpose of the Auth Mappings Page is to allow a DBA to monitor and review both the local and the global user authentication mappings. The list is read only and sourced from the RDB\$AUTH\_MAPPING and the SEC\$GLOBAL\_AUTH\_MAPPING tables.

It is only available for Firebird 3 or later.

## 4.11 The Database Repair Page

The purpose of the Database Repair Tab is to make available to the DBA Firebird's built in tools for Database Validation and Repair. These are also accessible using the Firebird *gfix* utility [18]. The DBA Tool uses the Services API for this purpose.

The functions available are:

- Database Sweep
- Online Validation
- Database Validation
- Kill Shadows

### 4.11.1 Database Sweep

This is invoked by selecting “Database Sweep” in the drop down box of Database Repair actions, and clicking on the “Run” button. The output of the service is written to the “Validation Report”.

Database Sweep is described in more detail in 3.6.4.

### 4.11.2 Online Validation

The Online Validation Service was introduced in Firebird 3 in order to permit a limited set of validation activities while the database is in normal use. When this service is selected in the drop down box of Database Repair actions, a list of database tables also appears to the left of the page. This allows the DBA to select a subset of the available tables for validation and which may be useful in reducing the time taken for the activity with large databases by focusing on suspected problems (see Illustration 19).

Click on “Run” to start Online Validation. The output of the service is written to the “Validation Report”.

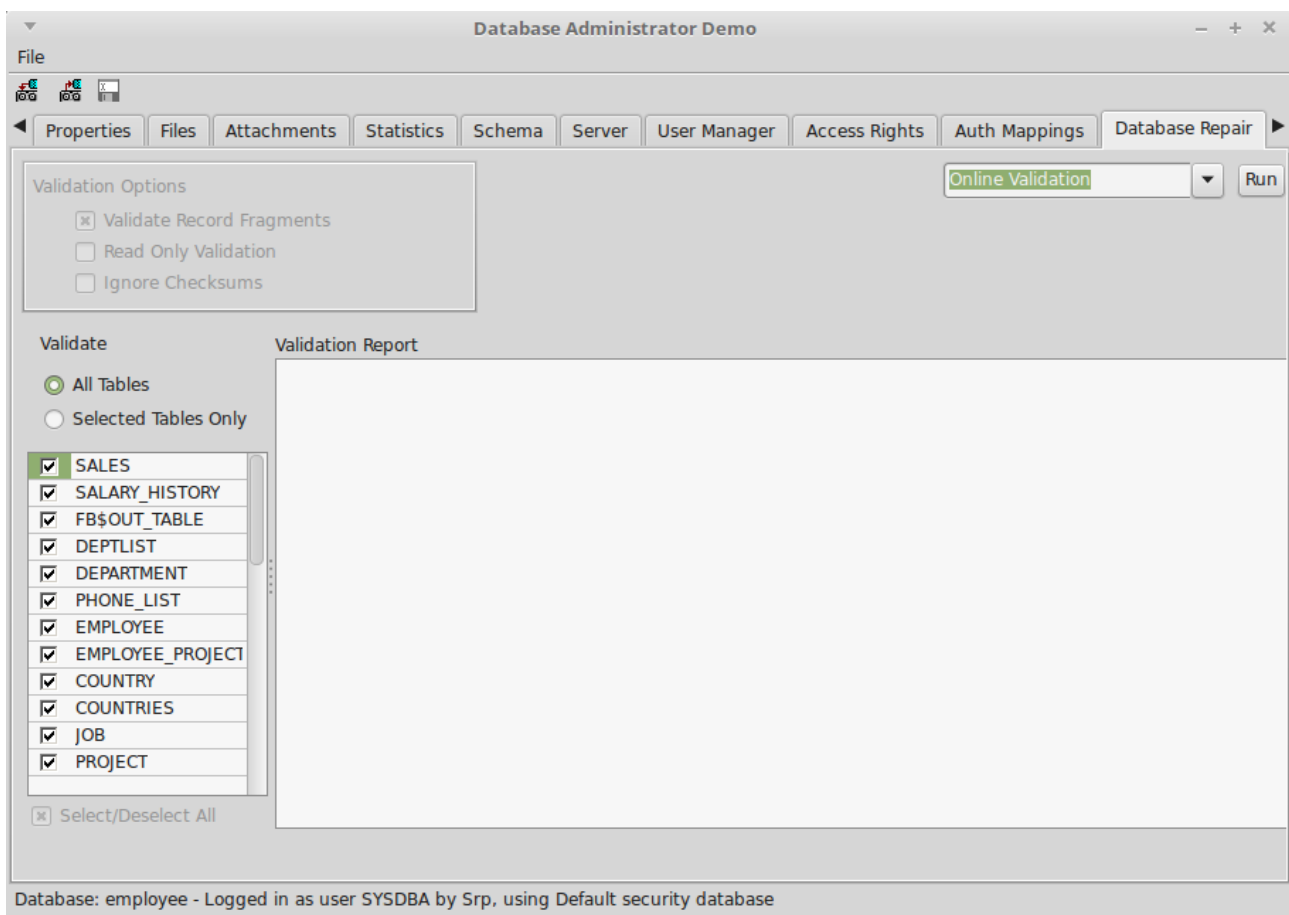


Illustration 19: Online Validation Example

### 4.11.3 Database Validation

Database Validation (and Repair) is more comprehensive than Online Validation, and is described as follows in the *gfix* documentation [18]:

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.

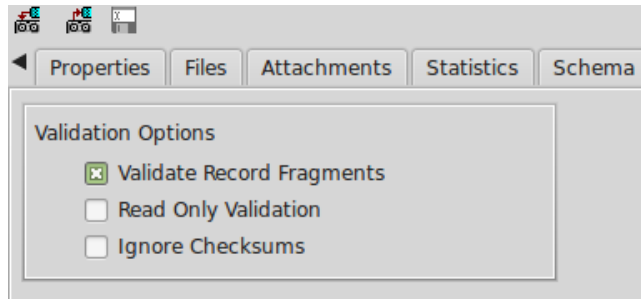
When Database Validation is selected in the drop down list of Database Repair actions, the “Validation Options” are enabled.

These allow the DBA to specify in addition to the basic action which:

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.

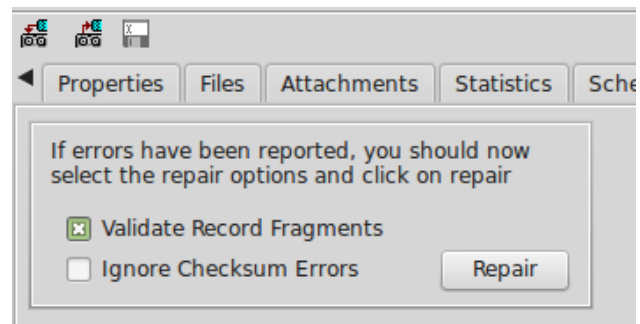
- **Validate Record Fragments:** using this option will validate, report and update at both page and record level. Any corrupted structures etc will be fixed.
- **Read Only Validation:** a read only validation simply reports any problem areas and does not make any changes to the database.
- **Ignore Checksums:** 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 this option.



**Illustration 20: The Phase One Validation Options**

Click on “Run” to start Database Validation (corresponds to *gfix* being run with the -v option). The output of the service is written to the “Validation Report”. If any errors are reported, the Phase Two “Database Repair” action may be performed.

The list of available options will have changed to “Validate Record Fragments” and “Ignore Checksums” only. Either or none may be selected. Click on the “Repair” button to perform the database repair function (corresponds to *gfix* run with the -m option).



**Illustration 21: Database Repair Options**

#### 4.11.4 Kill Shadows

This is a simple command that removes any unavailable database shadows. To use this function, select it from the drop down list and click on the “Run” button.

This function corresponds to the *gfix -kill* command.

## 4.12 The Limbo Transactions Page

The purpose of this page is to permit a DBA to perform Limbo Transaction recovery when the database is used as part of a two-phase commit over multiple databases. This function is described in detail in 3.6.3.

## 4.13 How It Works

This DBA Tool uses both the Database API and the Services API to communicate with the Firebird Server with the Database API being the primary means of communications. The program is structured into a “Backend” Data Module and a “Front End” form.

All operations are concentrated in the backend while the front end is largely concerned with presentation. This approach means that the backend could be readily integrated with alternative front ends in different user applications.

### 4.13.1 Database Connections

A Database Connection may be established in two ways:

1. At program start or when the DBA selects File->Open Database.
2. As an internal function when an action (e.g. Database Validation) requires that the Database connection is closed before the action is run through the Services API. In this case, the database connection is automatically re-opened after the action has been completed with the user being aware that this has happened.

In the former case, a Login Dialog is always displayed allowing the user to select the database and enter an appropriate user name and password. The program loops until successful login or the user cancels.

In the second case, no login dialog is displayed and the previously entered user credentials are used to connect to the database. Indeed, it is the existence of user credentials (user name and password) that determines whether or not a login dialog is displayed. The user credentials are cleared before case 1 above is invoked, but not before case 2.

### 4.13.2 Service API Login

It is normally necessary to connect to the Services API shortly after case 1 above of Database Connection established. In this case the same user credentials are used.

When an alternative security database is in use (see 2.4.1.4) then the **expected\_db** option needs to be set when connecting to the Services API. This is where having the database connection as the primary connection has advantages. When it comes to establishing the Services API connection, the Database Connection can be used to determine if an alternative security database connection is in use (see the MON\$DATABASE table) and, if so, the **expected\_db** parameter is set to the current database name. There should be no need to handle an *isc\_sec\_context* EIBInterBaseError exception (see 3.8.6).



## Appendix A. Potential Third Party Security Plugins

### A.1 Use of SRP for Database Key Management

This appendix explores the idea that a simple addition to SRP, together with the use of public key encryption could be used to simplify management of the keys used for database encryption. The plugin architecture allows for third party developed authentication modules and hence it is reasonable to speculate on how a third party might extend Firebird's SRP implementation. This section is such a speculation.

Public key encryption uses a key pair (the public and the private key) that are mathematically related such that data encrypted with the public key can only be decrypted with the private key and vice versa. The latter case is used for (e.g.) digital signatures, while the former can be used in (e.g.) key distribution.

The scheme envisaged here is that each client generates a public/private key pair. They retain the private key and keep it secure. The public key is freely distributable and is passed to a Database Administrator (DBA). The same DBA generates the database encryption key for server side encryption and separately encrypts it with each client's public key. For each client, the database encryption key encrypted with their public key is saved in the server's security database as part of the entry for the client's user name – perhaps as a user attribute tag.

Although the database encryption key has now been saved (maybe several times) on the server it is protected from an attacker as it is itself encrypted and it may be considered to be computationally infeasible to decrypt it without access to the associated user's private key.

Assuming a standard SRP implementation, when SRP returns its verifier M2, it has already verified the client and thus could also pass the version of the encrypted database key encrypted using the client's public key to the client alongside the M2 verifier.

When the client receives M2 plus the encrypted key, it may then authenticate the server and decrypt the database encryption key using its private key and return the unencrypted key to the server using the same protocol that the keyholder plugin would use and protected by “over the wire” encryption. The database encryption key is now available for the server to access the user database.

The benefits of this approach are that:

- The database encryption key no longer needs to be distributed to each client removing potential vulnerabilities in both distribution and storage. Its existence on the client is transitory and is discarded almost as soon as it is decrypted.
- It becomes much easier to revoke access to a database encryption key (at the same time as deleting a user's entry in the security database), or to change the database encryption key (only the security database's entries have to be updated – there is no separate need to distribute the updated key to each client).
- The client offers a second proof of identity in being able to decrypt the database key, where the first proof is the user password. This is because only the client that possesses the corresponding private key could have decrypted the database encryption key, and this could be used as the basis of two factor authentication.

- The same approach could be used to distribute client side encryption key(s) to users. Rather than just containing a single key, the information passed with M2 may be regarded as an encrypted package of security information for the client.

As with all public/private key systems, the DBA must be confident that they are using the correct public key for each client otherwise an attack vector opens up.

## A.2 An SSL/TLS Plugin for Firebird

Note: the term SSL/TLS is used above because of familiarity. However, the Secure Sockets Layer (SSL) protocol is now deprecated [16] in favour of Transport Layer Security (TLS) [17] and TLS is used here from now on.

The benefits of using TLS could be:

- Server authentication using X.509 certificates
- Client authentication and identification using X.509 certificates.
- Built-in “over the wire encryption” using many different encryption protocols.
- May be integrated with SRP (see [13] and [15]) and with the SRP extension suggested in A.1.
- The Public Key Infrastructure (PKI) supporting the X.509 certificates could also be used to verify the public key provided by a client to a DBA when the SRP extension suggested in is applied.
- May provide a non-repudiable proof of client authentication.

X.509 certificates are signed by a trusted third party – the Certification Authority (CA). Authentication that involves a trusted third party is potentially strong authentication, especially if an additional proof is provided, such as a password, when it may be considered a form of “two-factor authentication”. In the case of the client, the X.509 certificate may both identify and be limited to a single named user.

### ***TSL Client Authentication***

A TLS provider plugin could be configured to identify and authenticate clients either:

- By verifying a digital signature produced by the client and by verifying the client's public key with the client's X.509 certificate (which includes the client's user name), or
- By protecting a more traditional username/password exchange (with no client certificate required), or
- By demanding both of the above in order to gain a form of “two factor authentication”.

In the first bullet, the server may save the digital signature in order to retain a non-repudiable proof of user authentication.

### ***TLS Server Authentication***

A client may likewise authenticate a server by verifying a digital signature produced by the server and by verifying the server's public key with the server's X.509 certificate.



TLS requires that the server is provided with both a CA signed X.509 certificate (which is used to authenticate its public key) and that it also has a corresponding private key. This must be protected from an attacker either by:

- (a) Providing a secure path for a System Administrator to pass the server's private key to the Firebird Server TLS plugin, or
- (b) Saving the private key on the server in a passphrase encrypted file and, again, providing a secure path for a System Administrator to pass the passphrase to the Firebird Server TLS plugin.

The need for a secure path between the System Administrator and the Firebird Server TLS plugin is a common requirement.

Note: passphrase protection of a server key is an approach often taken when X.509 certificates are installed on web servers in support of "https"<sup>49</sup>.

In practice, whatever scheme is chosen to access and load the server's X.509 encryption key into memory could also be extended to support security database encryption (assuming that such a capability exists) by loading the security database(s) encryption key at the same time. They might (e.g.) be encrypted using the same passphrase. This could even extend to loading the encryption keys for user databases.

### ***TLS-SRP***

SRP has been integrated with TLS **[13]** and the SRP extension suggested in A.1 could also be added to this scheme. This would give the additional benefits that:

- The client's TLS private key may also be used to decrypt both server side and client side database encryption keys – as provided in the security information package saved in the security database and passed to the client alongside the SRP M2 verifier.
- The client's X.509 certificate can be used by a DBA to mitigate any risks from misidentification or impersonation of a client's public key when generating the per client encrypted database encryption keys.
- It should be possible to construct an authentication scheme that offers both authentication using a third party proof and two-factor authentication.

### ***With an Embedded Security Database***

TLS allows a client to authenticate a server and similarly a server to both authenticate and identify a client without requiring access to a security database. Access to the security database is only required for an additional client authentication step in a two factor authentication scenario. It would thus be feasible under TLS to embed the security database in an encrypted user database and for the client to provide the database encryption key to the server, after the client has authenticated the server but before the second authentication step.

However, this still means that the database encryption key has to be distributed to and stored on each client with the attendant vulnerabilities.

### ***With a Separate Encrypted Security Database***

On the other hand, a TLS-SRP with the SRP improvement suggested in A.1 is equally workable. A TLS implementation necessarily requires a secure pathway for a System Administrator to load the TLS server key and which also could be used to load the encryption keys for a separate and encrypted security database.

There is thus probably no great advantage with TLS in embedding the security database in a user database (as well as several downsides) compared with encrypting a separate security database, and the potential use of TLS-SRP provides a good reason not to embed the security database in an encrypted user database.

**Notes**

- 1 TCB = Trusted Computing Base i.e. the security-relevant portions of a system.
- 2 This is described in the section "Security->Over the Wire Connection Encryption" in the Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf). Note that an external "plugin" is used which permits extension to encryption algorithms other than the default ARC4. An ephemeral session key needs also to be agreed and this is usually a function of the user authentication mechanism.
- 3 Metadata access controls were introduced in Firebird 3. Prior releases allowed any user to create an object, whilst limiting changes to an object's owner or a "superuser".
- 4 The Firebird 2.5 release notes describe the RDB\$ADMIN role as something that can be granted to a user on a per database basis and/or to the security database. At the database level, it is granted the same as any other role using a GRANT statement (e.g. GRANT RDB\$ADMIN TO ALICE). However, an entirely different syntax is used for the security database (e.g. ALTER USER ALICE GRANT ADMIN ROLE"). See RDB\$ADMIN Role in Firebird 2.5.8 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-2.5.8-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-2.5.8-ReleaseNotes.pdf)
- 5 Firebird 3 has also added the SQL standard "GRANTED BY" clause to the Grant statement. This allows a SYSDBA or Database Administrator to specify a different user as the "grantor" of the access right.
- 6 Prior to Firebird 3, the System Tables used to hold a database's metadata were both publicly readable and writable by a System Administrator. In Firebird 3, they become read only, with explicit permissions controlling which users can update the metadata and which metadata objects they can update (through DDL statements). (e.g. making "Create Table" and "Create Procedure" separate assignable privileges". See "User Privileges for Metadata Changes" Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf).
- 7 It was, of course, possible to use an externally encrypted file system.
- 8 This is an often overlooked aspect of file system security. If the file system path to a database or configuration file can be modified then this introduces all sorts of ways in which an attacker can make mischief. For example, by diverting the path to the security database to point to a compromised version.
- 9 Prior to Firebird 3, this file was called "aliases.conf".
- 10 The definition of a "cloud server" is itself a bit cloudy and is probably best assumed to be a marketing term meaning that you know not where your data is being stored, how long it will be stored for or who has access to it. Provided that you can live within these parameters, cloud storage can be very efficient and cost effective, but its use for sensitive data is more problematic and may demand some sort of encryption under the control of the data owner/controller and not the cloud service provider.
- 11 See section "Database Encryption" in the Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf)
- 12 The assumed model here is that a System Administrator is able to contact the server using a secure channel (e.g. using SSH) and provides the key to the Firebird Server's encryption plugin, which then keeps it in memory for use by authenticated users. When the Firebird server stops, the key is lost and the database is inaccessible until the System Administrator once again uses the secure channel to provide the key.
- 13 The release notes do not actually make a recommendation. However, the impression given is that this is the model that the authors had in mind.
- 14 A poorly protected server is always vulnerable to server masquerade (through theft of the server side authentication data) or interception of security information on the server itself, or some other vulnerability.

- 15 It should be recalled that a major DAC requirement is that “The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user”. It is very easy to overlook the importance of protecting the security database and, in the absence of database encryption, that file system protections have to be relied upon to do this. This includes both limiting read and write access to authorised users only and ensuring that the database's file system path is protected from unauthorised modification. Otherwise a malicious user could replace the security database with a compromised version. All directories in the security database's file system path must restrict write access to authorised users only.
- 16 Firebird's vulnerability to brute force attacks has been significantly reduced over successive releases. For a discussion of this subject see the “Details of the Security Changes in Firebird 2” in the Firebird 2.1.7 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/html/rlsnotes217.html](https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes217.html)
- 17 This subject is discussed at length in the release notes. See section “Details of the Security Changes in Firebird 2.0” in Firebird 2.0.7 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/html/rlsnotes207.html](https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes207.html).
- 18 If this feature is used then the DBA needs to be confident that the user authentication tables are adequately protected from both read and write access by normal users. This includes preventing normal users from making a remote backup of the database (including authentication tables). See “Location of User Lists” in Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf)
- 19 In this case, the client will need to provide the encryption key to the server before the client is authenticated. However, in order to avoid keys being inadvertently disclosed to an attacker, the user authentication protocol must permit the server to be authenticated to the client prior to communication of the encryption key. The current set of Firebird user authentication plugins do not appear to meet this requirement.
- 20 See section on “Creating an Alternative Security Database” Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf).
- 21 For a list of all parameters available for use in *databases.conf* and the file's syntax see the section “Per-database Configuration” in Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf).
- 22 On a Debian derived system such as Ubuntu, *isql* is usually called *isql-fb*.
- 23 This is described in the file “doc/README.services\_extension” provided with the Firebird 3 source code.
- 24 In modulo arithmetic, exponentiation can be performed in a deterministic time. However, the inverse operation i.e. computing the logarithm of a number is believed to be computationally infeasible especially when the base and modulus are carefully chosen. i.e. while  $A = g^a \bmod N$  may be computable, the inverse  $a = \log_g A \bmod N$  is not.
- 25 The double hash is a Firebird variation.
- 26 This is stated in the sections “Increased Password Length” and “Specifications for the Key” in the Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf), and appears to be confirmed by source code review. SHA-1 is also specified in the original SRP RFC [10]. However, later descriptions of SRP only refer to a secure hash algorithm. This is for good reason, SHA-1 is now known to be vulnerable to collision attacks (a Google search for “sha1 vulnerability” is instructive – see also [20]). How seriously such attacks affect its use in SRP is not yet clear. However, it would be desirable for Firebird to implement a more robust secure hash algorithm in the next release.
- 27 This is another Firebird deviation. SRP originally specified  $M1 = H(A, B, S)$ , while the SRP website now recommends  $M1 = H(H(N) \mathbf{xor} H(g), H(I), s, A, B, K)$ . It is not known why the Firebird implementation

uses exponentiation instead of xor. As both  $N$  and  $g$  are well known and fixed, both variations result in a global constant for the first term in the hash.

- 28 In the "Over the wire" Connection Encryption section of the Firebird 3 release notes, it is stated that "RC4 uses a symmetric key which can have any length, while the key produced by SRP has a length of 20 bytes. That key is a SHA-1 hash on SRP's session key and some other SRP-related things, such as user name". See Firebird 3.0.3 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf)
- 29 A Google search for "sha1 password cracker" is instructive.
- 30 As the discrete logarithm problem is the primary defence against password cracking, it is not clear why Firebird saw the need for additional protection using a double hash when computing the value of  $x$ .
- 31 A Digital Signature as defined by [21] has two steps associated with its generation. The first is the use of a cryptographic hash function over the data to be signed and the second step involves a cryptographic key and signing function. The second step ensures that the digital signature is non-repudiable. In SRP, only the first step is applied with the result that the signature cannot be considered to be non-repudiable.
- 32 A useful definition of Weak Authentication is provided by Jari Arkko, Pekka Nikander as "cryptographic authentication between previously unknown parties without relying on trusted third parties."  
(<https://www.ietf.org/proceedings/57/slides/enroll-4/enroll-4.ppt>)
- 33 The SRP authentication plugin is stated as having an effective password length of 20 bytes (Firebird 3.0.3 Release Note Plugins Q & A) – although as International Character Sets are also supported by Firebird 3 (e.g. UTF8), the actual effective password size in characters is not easy to quantify. It is also not clear as to where this restriction comes from and does seem rather similar to the (unrelated) length of a SHA-1 digest. The Firebird release notes later state (see section under heading "Increased Password Length) "Longer passwords can be used without restriction but there is a remote possibility of hash collisions between passwords that differ beyond the 20th byte. Just be aware of the possibility that any password longer than 20 characters password could share the same hash with some shorter password so, theoretically, they could be attacked using brute force". This seems an odd argument as SRP never hashes a password on its own, but always with something else. It would also be less of an issue if (e.g.) SHA-256 had been used instead of SHA-1 (see note above). Firebird 3.0.3 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf)
- 34 The Firebird 3 release notes do not explicitly state that Win\_Sspi cannot support over the wire encryption. However, a comment in the firebird.conf file states "If you need to use server plugins that do not provide encryption key (both Legacy\_Auth & Win\_Sspi) you should also turn off required encryption on the wire with WireCrypt".
- 35 See Configuration Parameter "Authentication" in the Firebird 2.1.7 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/html/rlsnotes217.html](https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes217.html).
- 36 See "Global Admin Privileges for Windows Administrators" in the Firebird 2.5.8 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-2.5.8-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-2.5.8-ReleaseNotes.pdf).
- 37 This is effectively enforced by the use of the username in all access rights that are part of each database's metadata. If you could change the username in the security database there would still be a need to also change the same username in every database to which the user has access, including backups.
- 38 This is also described in the file "doc/README.services\_extension" provided with the Firebird 3 source code.
- 39 Although deprecated, in Firebird 3 the services API is the only mechanism available for User Management in the somewhat unusual case where a normal user is given the Admin role in a security

database but does not have the RDB\$ADMIN role in any database that uses it. In this case, they are denied full use of the SEC\$USERS table and hence can only see a full list of users when using the Services API. Such a user has no way of accessing the user list in an alternative security database.

- 40 See section “New System Tables” in Firebird 3.0.3 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf).
- 41 See section “Security->SQL Features for Managing Access” in Firebird 3.0.3 Release Notes  
[https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf)
- 42 The upgrade procedure is described in the section “Upgrading a v.2.x Security Database” in the Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf).
- 43 This was introduced in Firebird 2.5. See section “New Extensions to EXECUTE STATEMENT” in the Firebird 2.5.8 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-2.5.8-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-2.5.8-ReleaseNotes.pdf).
- 44 The Firebird 3 Release Notes make clear in that mappings are there to support (a) when EXECUTE STATEMENT ON EXTERNAL DATA SOURCE requires some data exchange between database clusters using different security databases, (b) when server-wide SYSDBA access to databases is needed from other clusters, using services, and (c) with Windows trusted authentication, to relate the Windows security database with Firebird's.
- 45 Non-native English speakers may not be familiar with this use of the word “assume” where it means “take on or adopt” instead of its more normal semantic in a logical argument as “accept as true without proof”.
- 46 Rolenames follow the same naming rules as user names. However, “It is advisable to make the name of a role unique among user names as well. The system will not prevent the creation of a role whose name clashes with an existing user name but, if it happens, the user will be unable to connect to the database.” This is a quote from the “DDL Statements” section in the Firebird 2.5 Language Reference [https://www.firebirdsql.org/file/documentation/reference\\_manuals/fblangref25-en/html/fblangref25.html](https://www.firebirdsql.org/file/documentation/reference_manuals/fblangref25-en/html/fblangref25.html)
- 47 Copied from the Firebird 2.5 Language Reference [https://www.firebirdsql.org/file/documentation/reference\\_manuals/fblangref25-en/html/fblangref25.html](https://www.firebirdsql.org/file/documentation/reference_manuals/fblangref25-en/html/fblangref25.html) corrected and updated from the section “Privileges to Protect Other Metadata Objects” in the Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf).
- 48 See section “User Privileges for Metadata Changes” in the Firebird 3.0.3 Release Notes [https://www.firebirdsql.org/file/documentation/release\\_notes/Firebird-3.0.3-ReleaseNotes.pdf](https://www.firebirdsql.org/file/documentation/release_notes/Firebird-3.0.3-ReleaseNotes.pdf)
- 49 This is a common feature with the Apache webserver. When it uses an encrypted server key, the server must be manually started by a System Administrator who is then prompted for the passphrase. Once the key is loaded into memory, it is available for use until the server is stopped.