



MWA Software

Firebird Server Management using IBX

Issue 1.0,
Doc Ref: MWA/DBAdmin
28 February 2018

McCallum Whyman Associates Ltd

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
1 INTRODUCTION.....	1
1.1 REFERENCES.....	2
2 USING THE IBX SERVICES API FOR DATABASE ADMINISTRATION.....	3
2.1 RUNNING THE EXAMPLE.....	3
2.2 DATABASE BACKUP.....	4
2.3 DATABASE RESTORE.....	5
2.4 SERVER LOG.....	5
2.5 USER MANAGEMENT.....	6
2.6 THE “DATABASE” ACTIONS.....	7
2.6.1 Show Statistics.....	7
2.6.2 Validation.....	7
2.6.3 Limbo Transaction Resolution.....	7
2.6.4 Database Sweep.....	9
2.6.5 Database Shutdown.....	10
2.6.6 Bringing a Database Backup Online.....	10
2.7 USING AN ALTERNATIVE SECURITY DATABASE.....	11
2.8 HOW IT WORKS.....	11
2.8.1 Services API Login.....	13
2.8.2 IBX for Lazarus and the Services API.....	13
2.8.3 IBX 2.2 and the Services API.....	13
2.8.4 User Management.....	13
2.8.5 Limbo Transactions.....	14
2.8.6 Using Alternative Security Databases.....	14
2.9 SUMMARY.....	15
3 THE DBADMIN TOOL.....	17
3.1 RUNNING THE EXAMPLE.....	18
3.1.1 The RDB\$ADMIN role.....	18
3.2 DATABASE PROPERTIES.....	19
3.2.1 Database Backup.....	20
3.2.2 Database Restore.....	21
3.3 THE FILES PAGE.....	22
3.3.1 Adding a Secondary File.....	22
3.3.2 Shadow Sets.....	23
3.3.2.1 Adding a Shadow Set.....	23
3.3.2.2 Dropping a Shadow Set.....	24
3.4 THE ATTACHMENTS PAGE.....	24
3.5 THE STATISTICS PAGE.....	24
3.6 SCHEMA PAGE.....	25
3.7 THE SERVER PAGE.....	25
3.8 THE USER MANAGER PAGE.....	25
3.9 THE ACCESS RIGHTS PAGE.....	27
3.9.1 Stale Users.....	28
3.10 THE AUTH MAPPINGS PAGE.....	29
3.11 THE DATABASE REPAIR PAGE.....	29
3.11.1 Database Sweep.....	29
3.11.2 Online Validation.....	29
3.11.3 Database Validation.....	30
3.11.4 Kill Shadows.....	31
3.12 THE LIMBO TRANSACTIONS PAGE.....	31
3.13 HOW IT WORKS.....	32
3.13.1 Database Connections.....	32
3.13.2 Service API Login.....	32

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

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
3. Firebird 2.5.8 Release Notes
https://www.firebirdsql.org/file/documentation/release_notes/Firebird-2.5.8-ReleaseNotes.pdf
4. Firebird 2.5 Language Reference
https://www.firebirdsql.org/file/documentation/reference_manuals/fblangref25-en/html/fblangref25.html
5. Firebird Database Housekeeping Utility (gfix) -
<https://www.firebirdsql.org/pdfmanual/html/gfix.html>

2

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 3 uses virtual database tables wherever possible and the Services API only when it has too. The result is arguably a much “slicker” tool.

2.1 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.

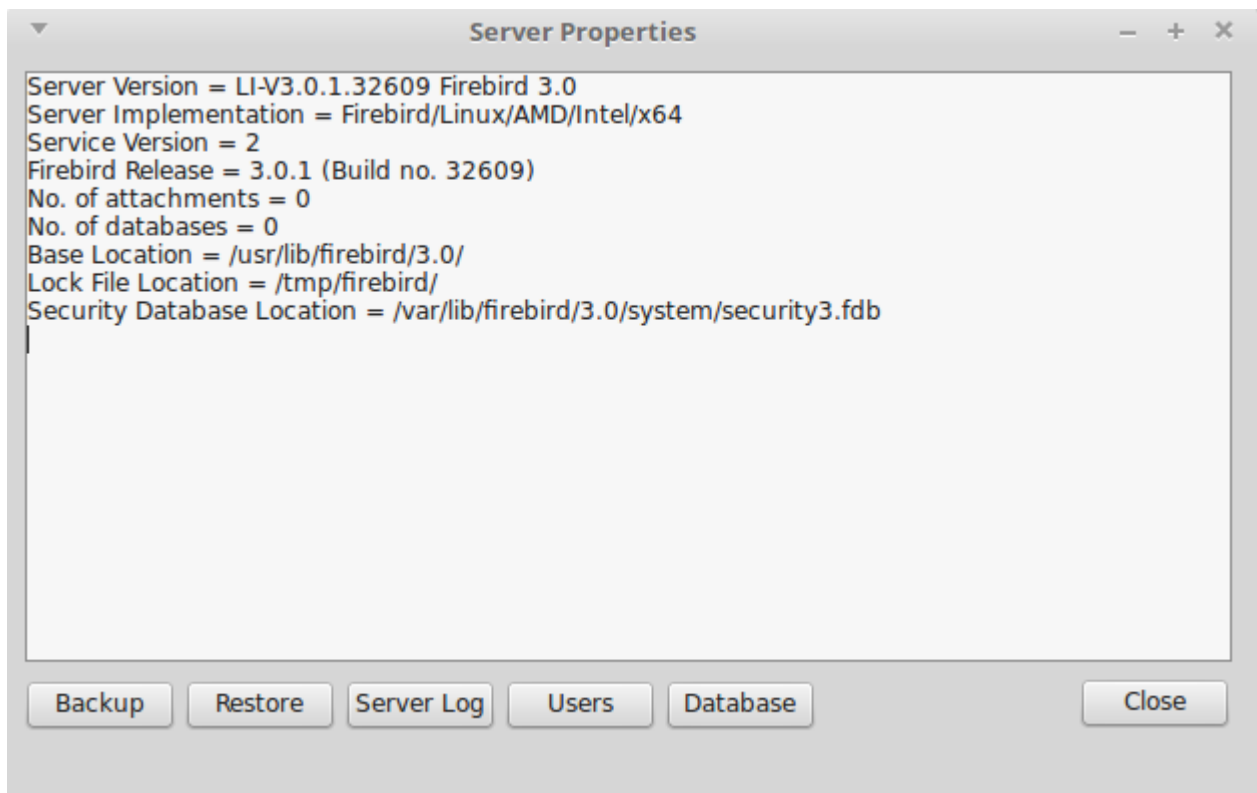


Illustration 1: Services API Example Main Form

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.

2.2 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.

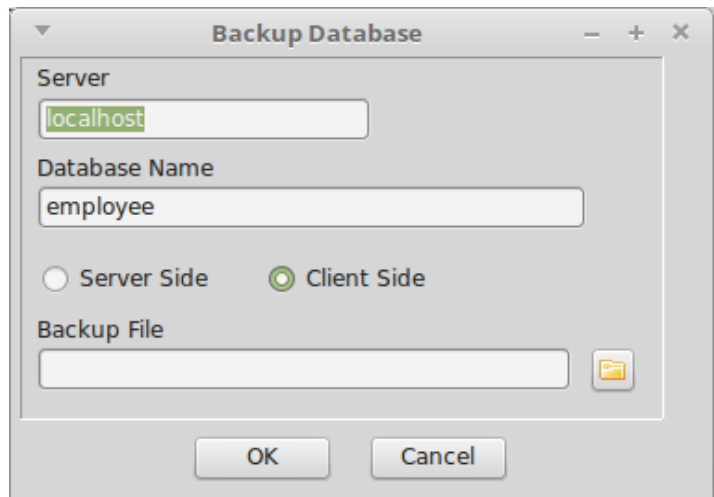


Illustration 2: Database Backup Dialog

2.3 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.

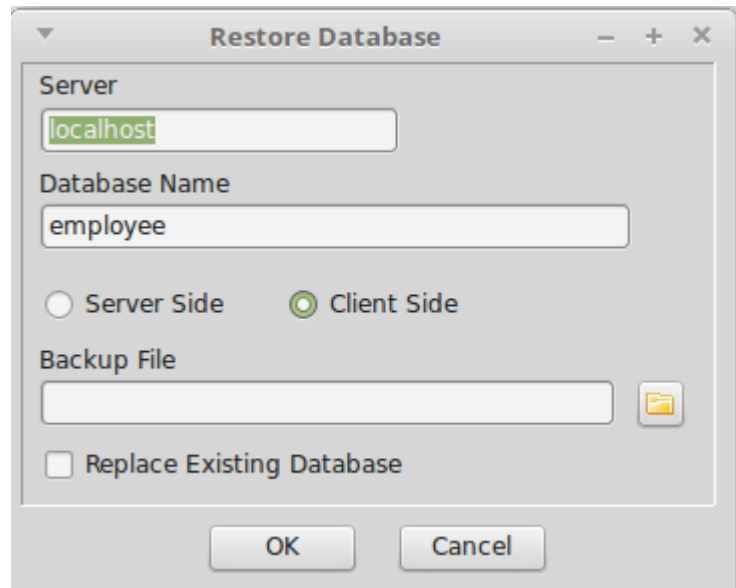


Illustration 3: Database Restore Dialog

2.4 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.

2.5 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 an 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 3.

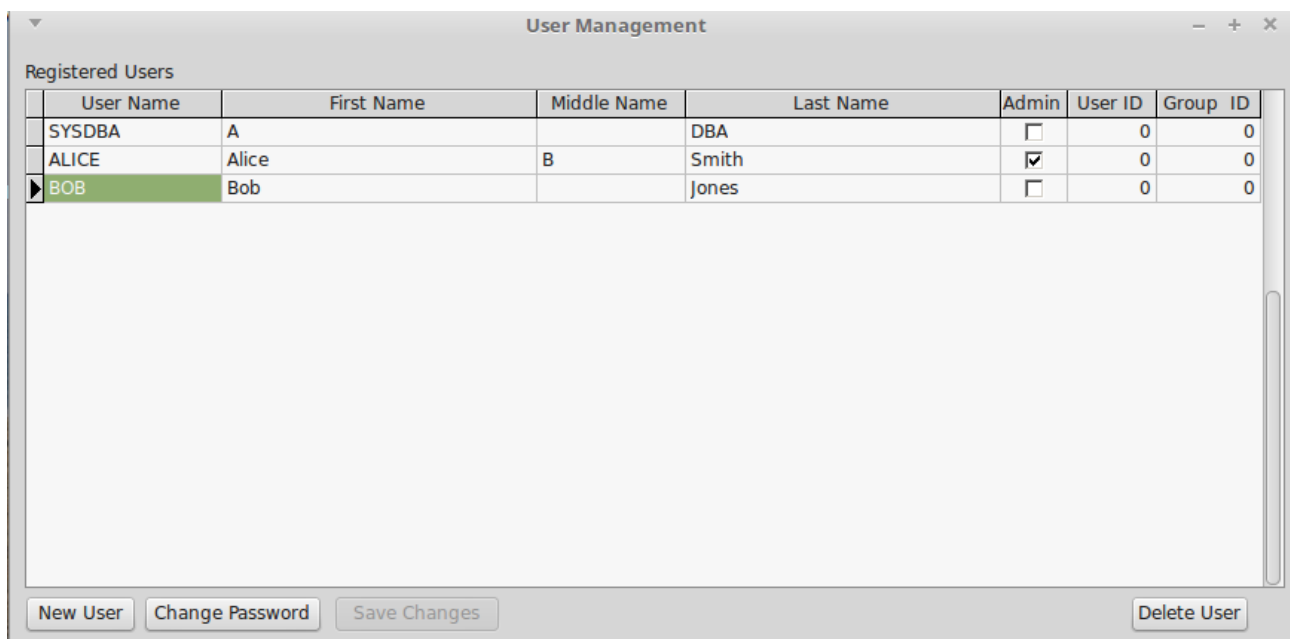


Illustration 4: User Management Dialog

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.

2.6 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

2.6.1 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.

2.6.2 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.

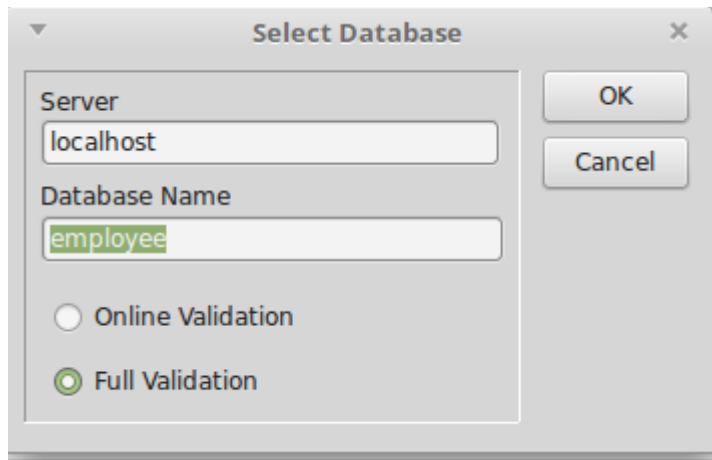


Illustration 5: Database Selection Dialog for Validation

2.6.3 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.

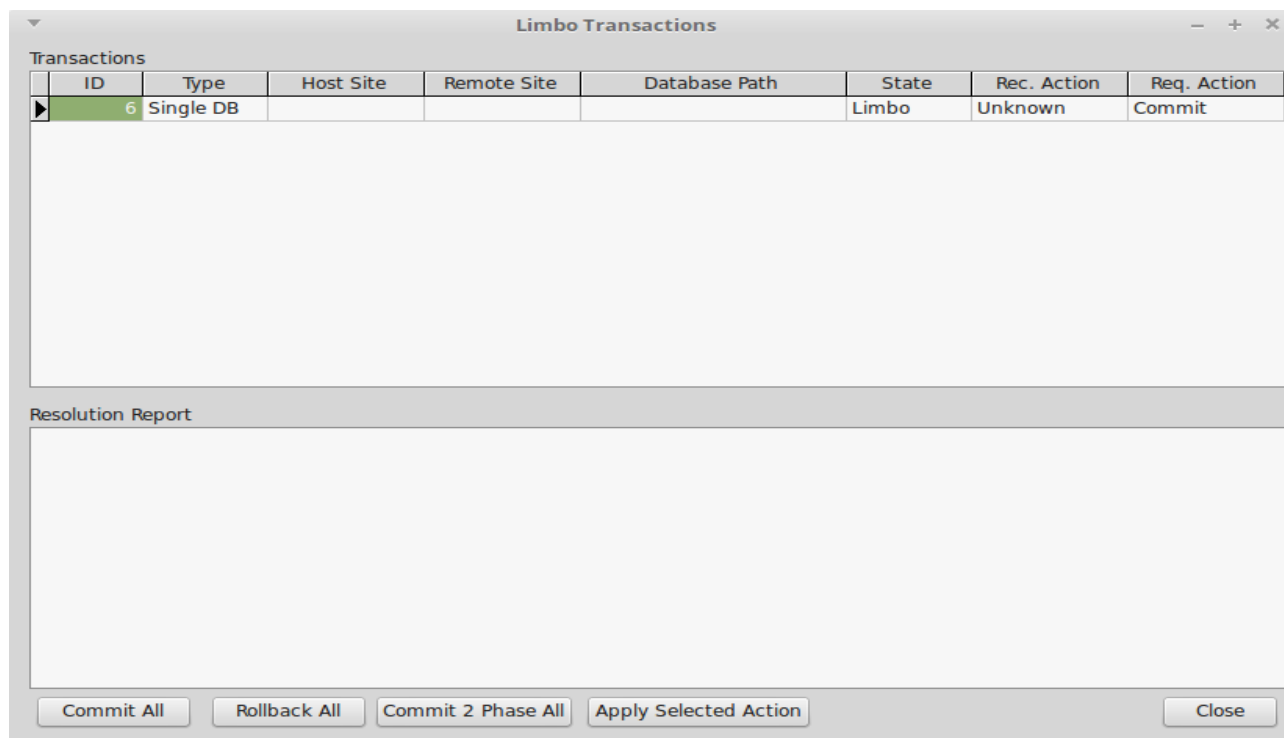


Illustration 6: Limbo Transaction Resolution Dialog

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.

2.6.4 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.

2.6.5 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.

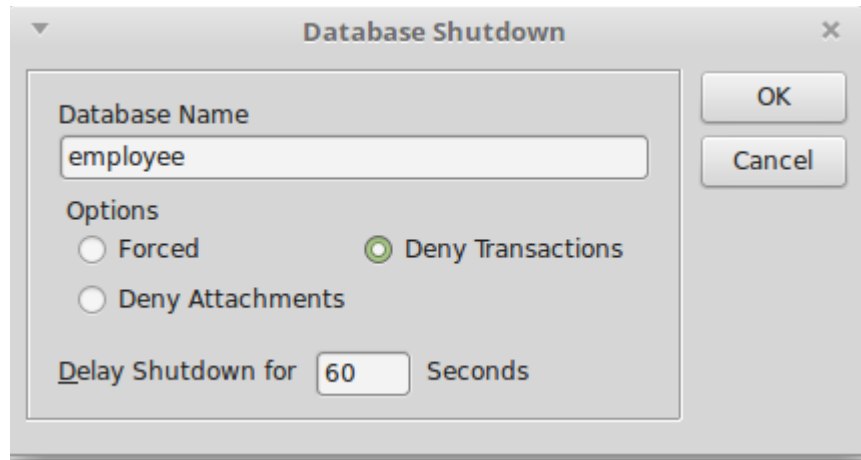


Illustration 7: Database Shutdown Dialog

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.

2.6.6 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 bring back online and the service is invoked. It always returns immediately to report success or failure (usually because the database was already online).

2.7 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.

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], 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.










2.8 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.

2.8.1 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.

2.8.2 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.

2.8.3 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.

2.8.4 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.

2.8.5 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.

2.8.6 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.

2.9 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.

3

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 Database Administration. 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.

3.1 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 3.1.1) 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.

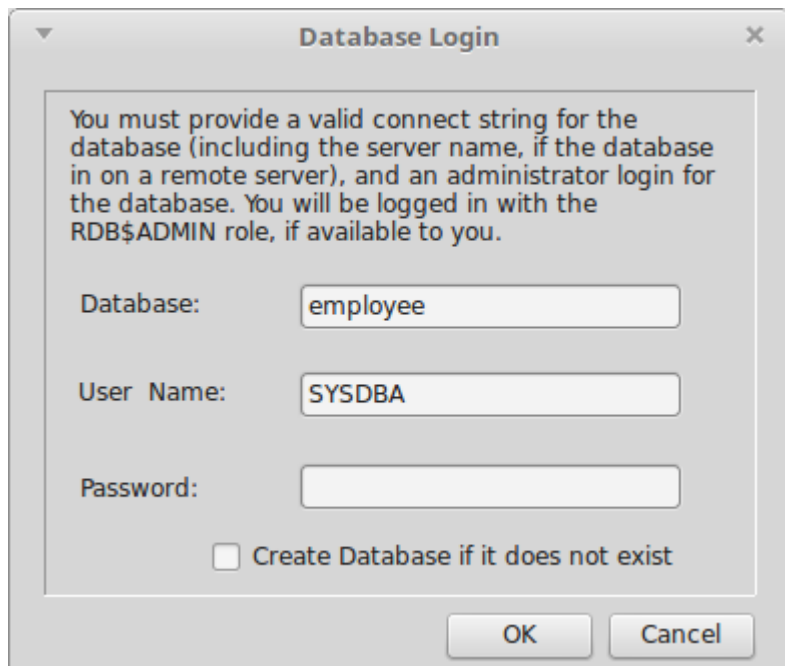


Illustration 8: Database Login Dialog

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.

3.1.1 The RDB\$ADMIN role

The RDB\$ADMIN role was introduced in Firebird 2.5 [3] 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 login or by using the “SET ROLE RDB\$ADMIN” statement.

3.2 Database Properties

The initial display shows the database properties.

The screenshot shows the 'Database Administrator Demo' window with the 'Properties' tab selected. The window displays various database configuration parameters and their current values.

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

Additional options (checkboxes):

- ☐ Read Only
- ☒ Online
- ☒ Synchronous Disk Writes
- ☐ Shadow Database
- ☒ Space Reserved for Backup Records
- ☐ Auto Admin Mapping

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

Illustration 9: Database Administration Example Application

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”.

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 2.6.5) and uses the `TIBConfigService`. The reverse action brings the database back online.
- Auto Admin Mapping allows Windows Administrators logged in using trusted authentication to automatically be granted the `RDB$ADMIN` role.

3.2.1 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.

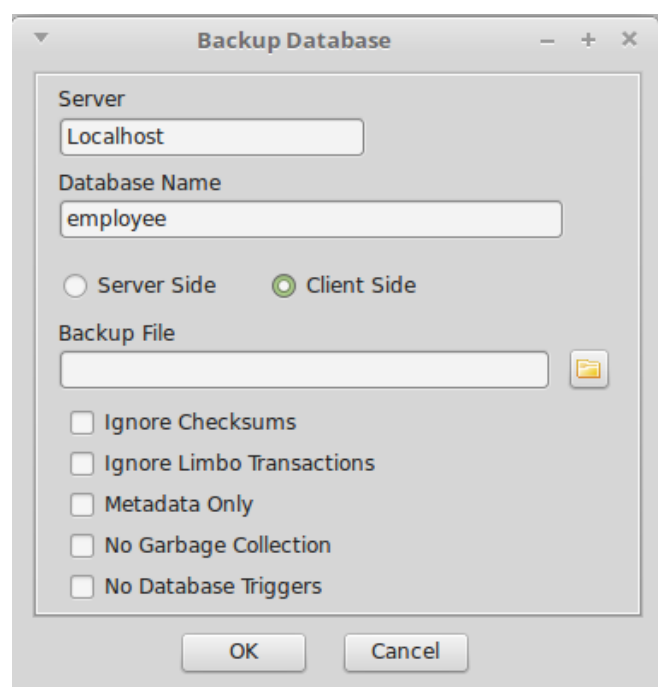


Illustration 10: Database Backup Dialog

3.2.2 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.

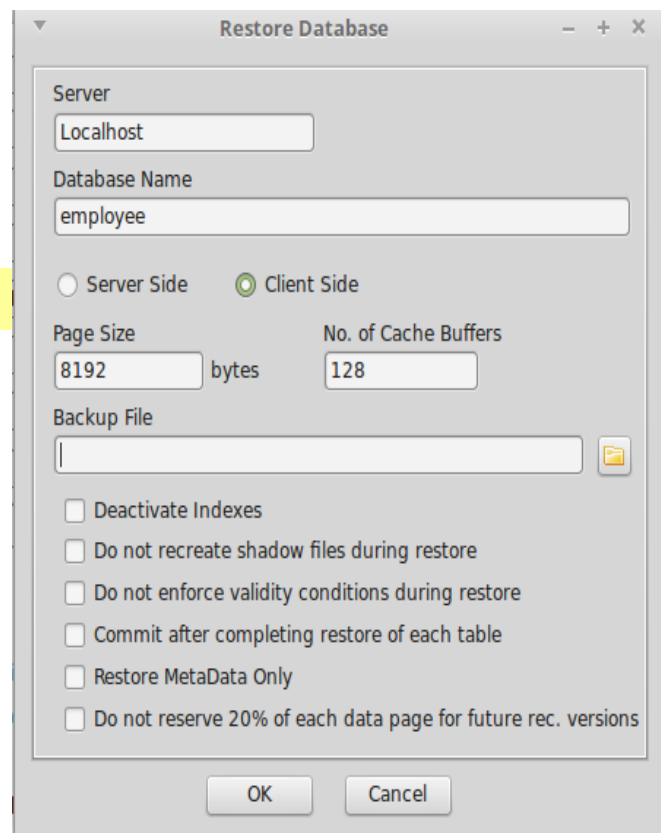


Illustration 11: Database Restore Dialog

3.3 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.

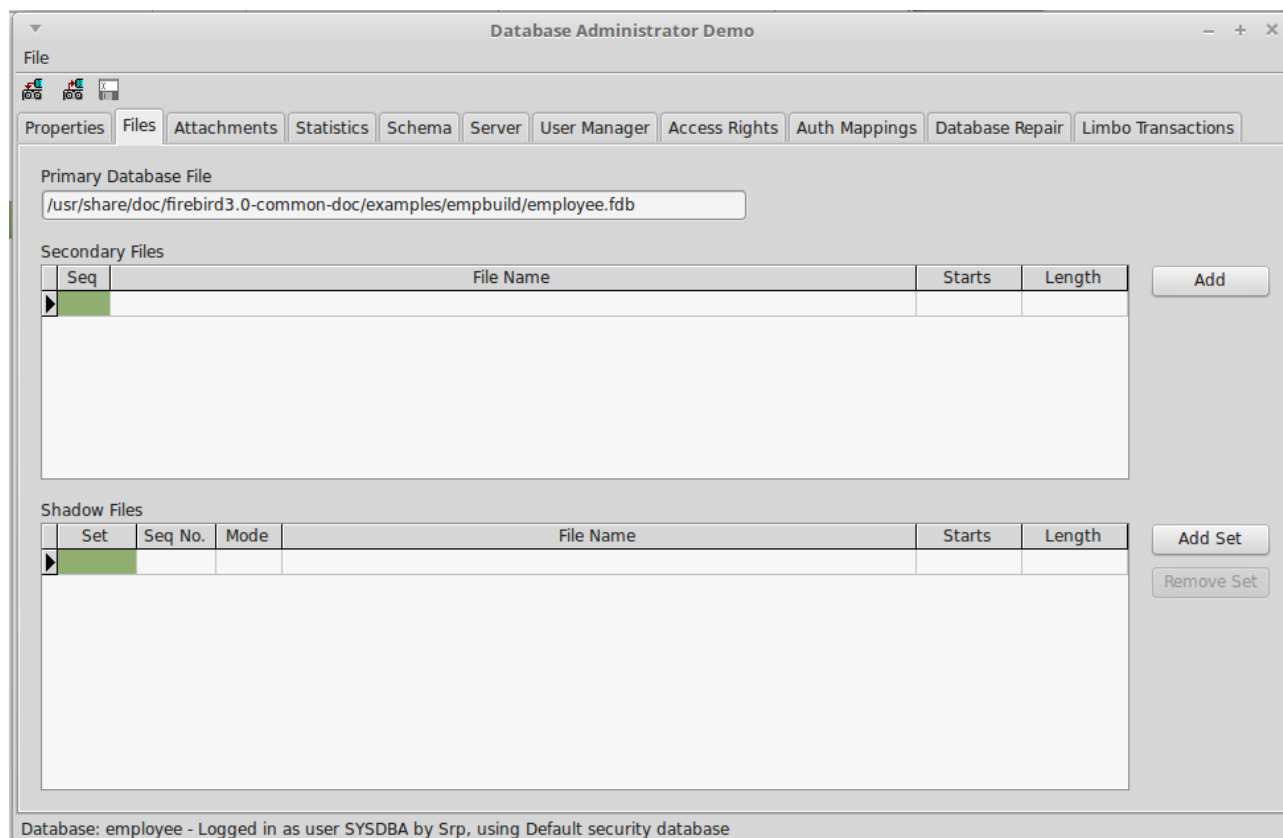


Illustration 12: The DBAdmin File Page

3.3.1 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.

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.

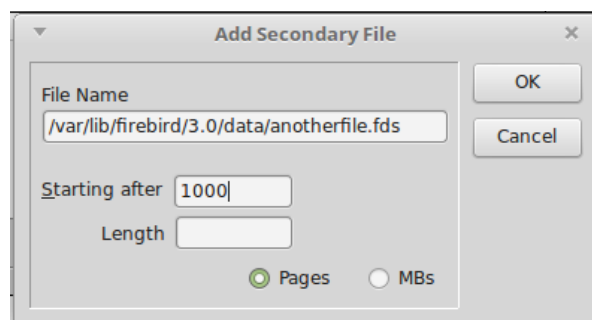


Illustration 13: Add Secondary File Dialog

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 [4] should be consulted for more information.

3.3.2 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.

3.3.2.1 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.

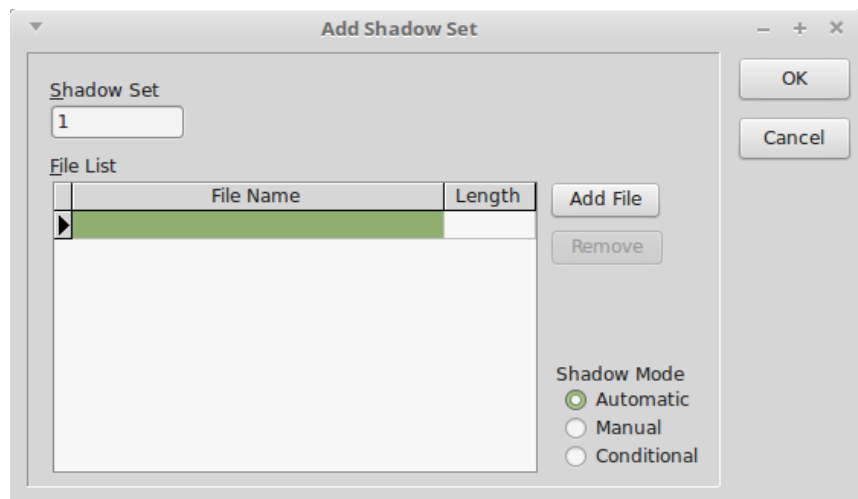


Illustration 14: Add Shadow Set Dialog

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.

3.3.2.2 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).

3.4 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.

3.5 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.

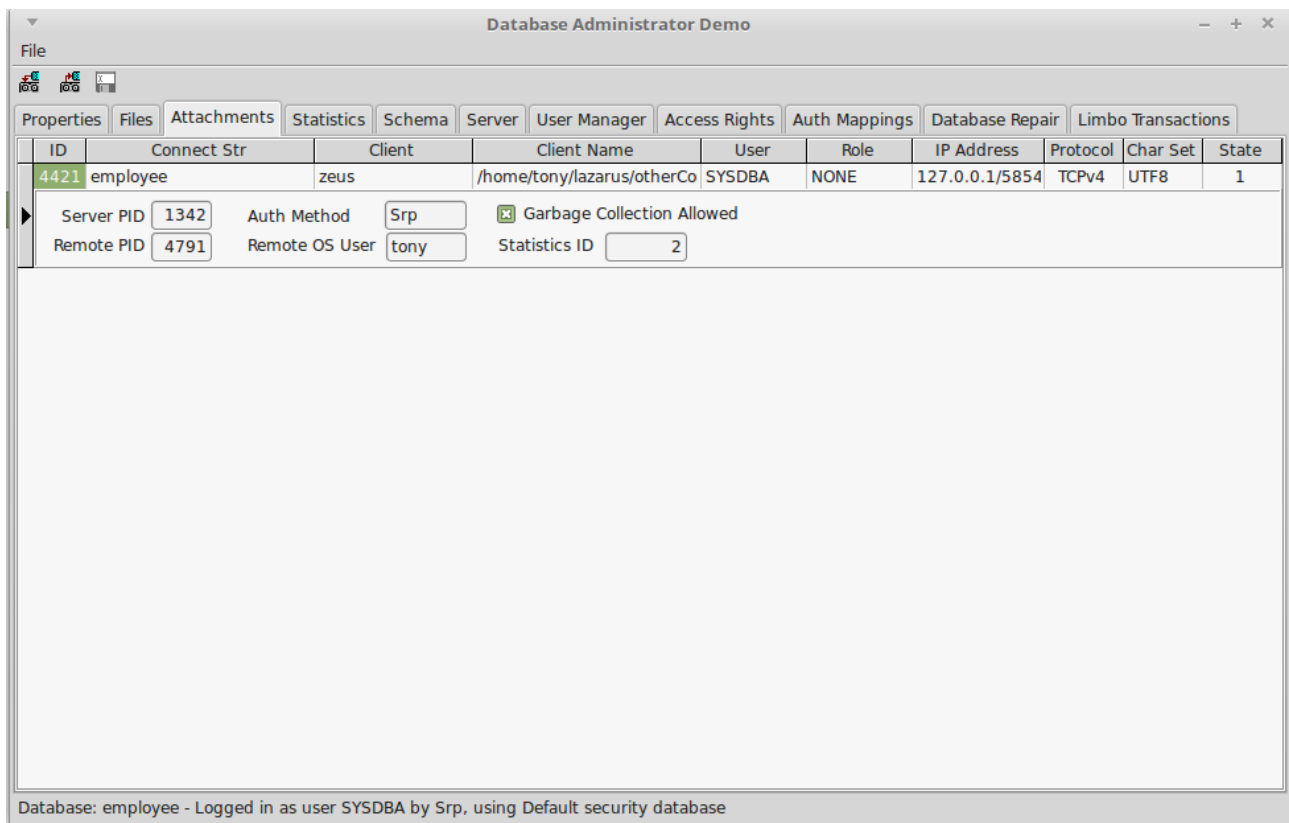


Illustration 15: Database Attachments Page

3.6 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.

3.7 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.

3.8 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.

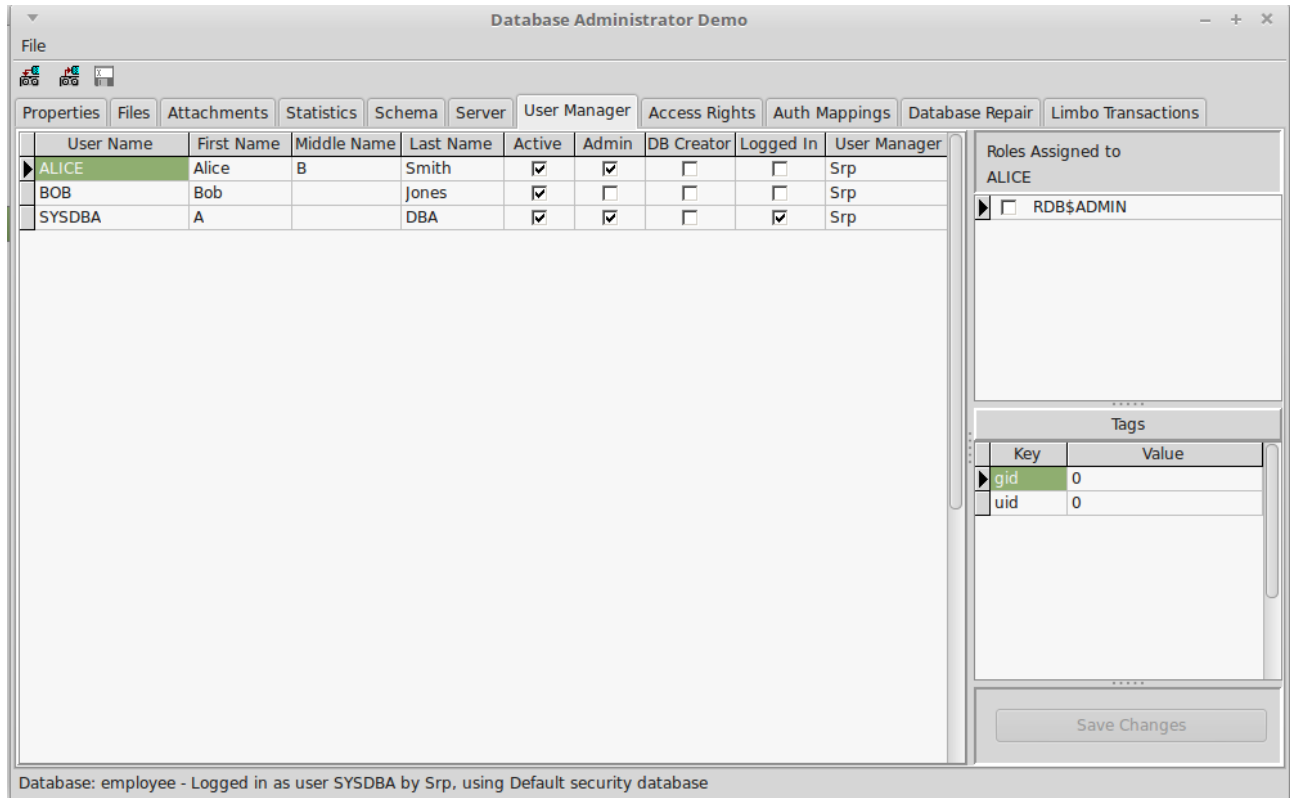


Illustration 16: The User Management Tab

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 2.8.4. 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 2.8.4.

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).

3.9 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.

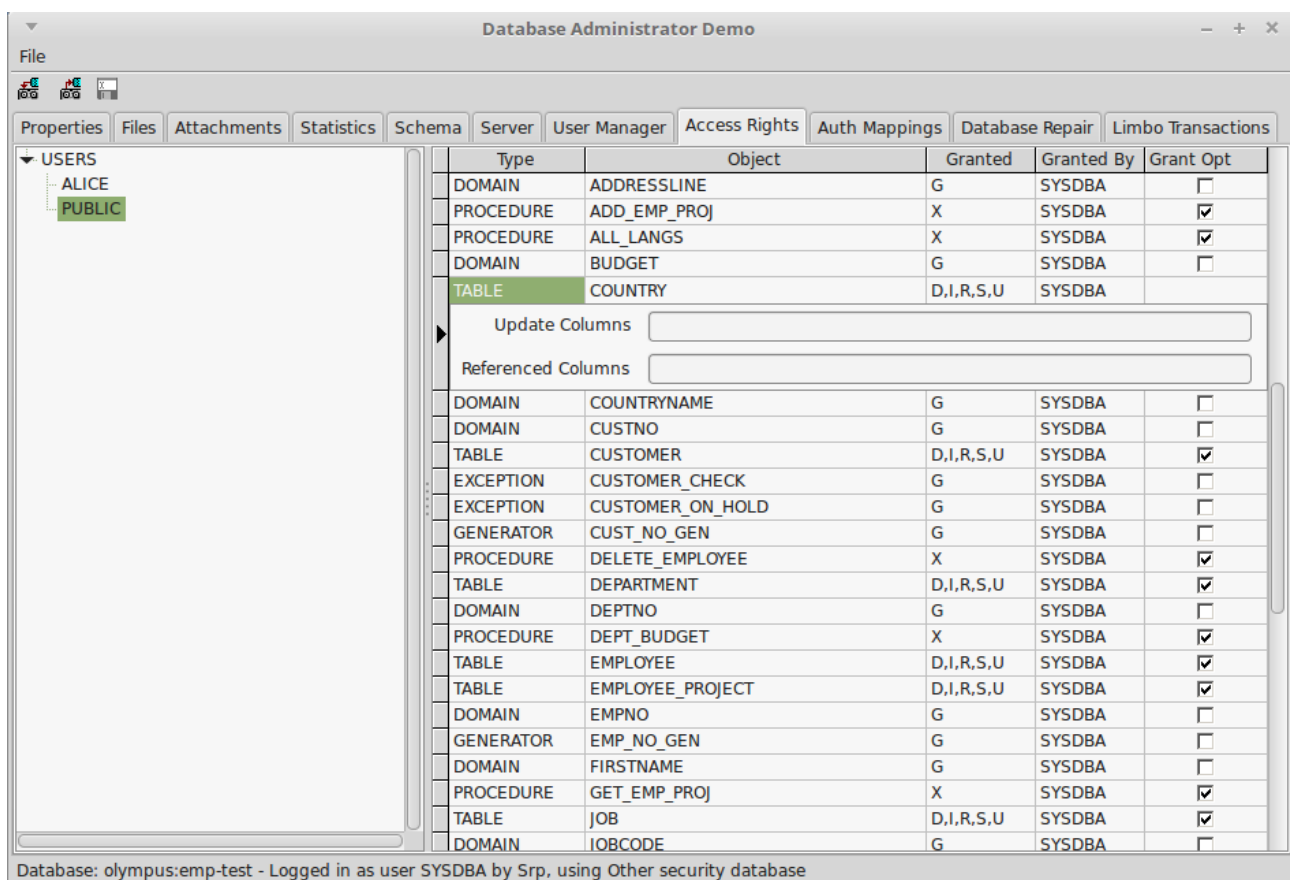


Illustration 17: Access Rights Page

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).

3.9.1 Stale Users

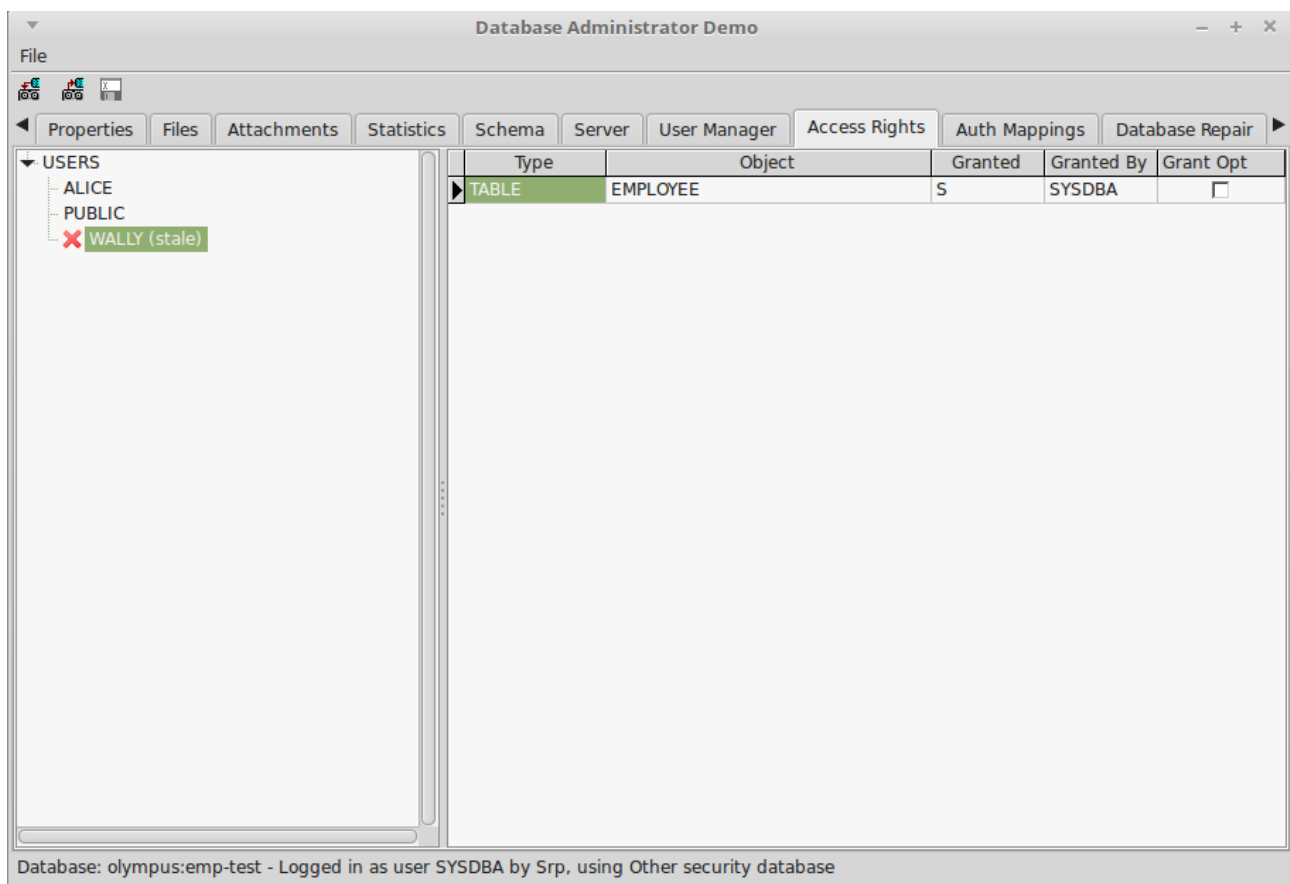


Illustration 18: State Access Rights Indication

The risk of “stale” user access rights existing in a database was identified in Error: Reference source not found. 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.

3.10 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.

3.11 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 [5]. The DBA Tool uses the Services API for this purpose.

The functions available are:

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

3.11.1 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 2.6.4.

3.11.2 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”.

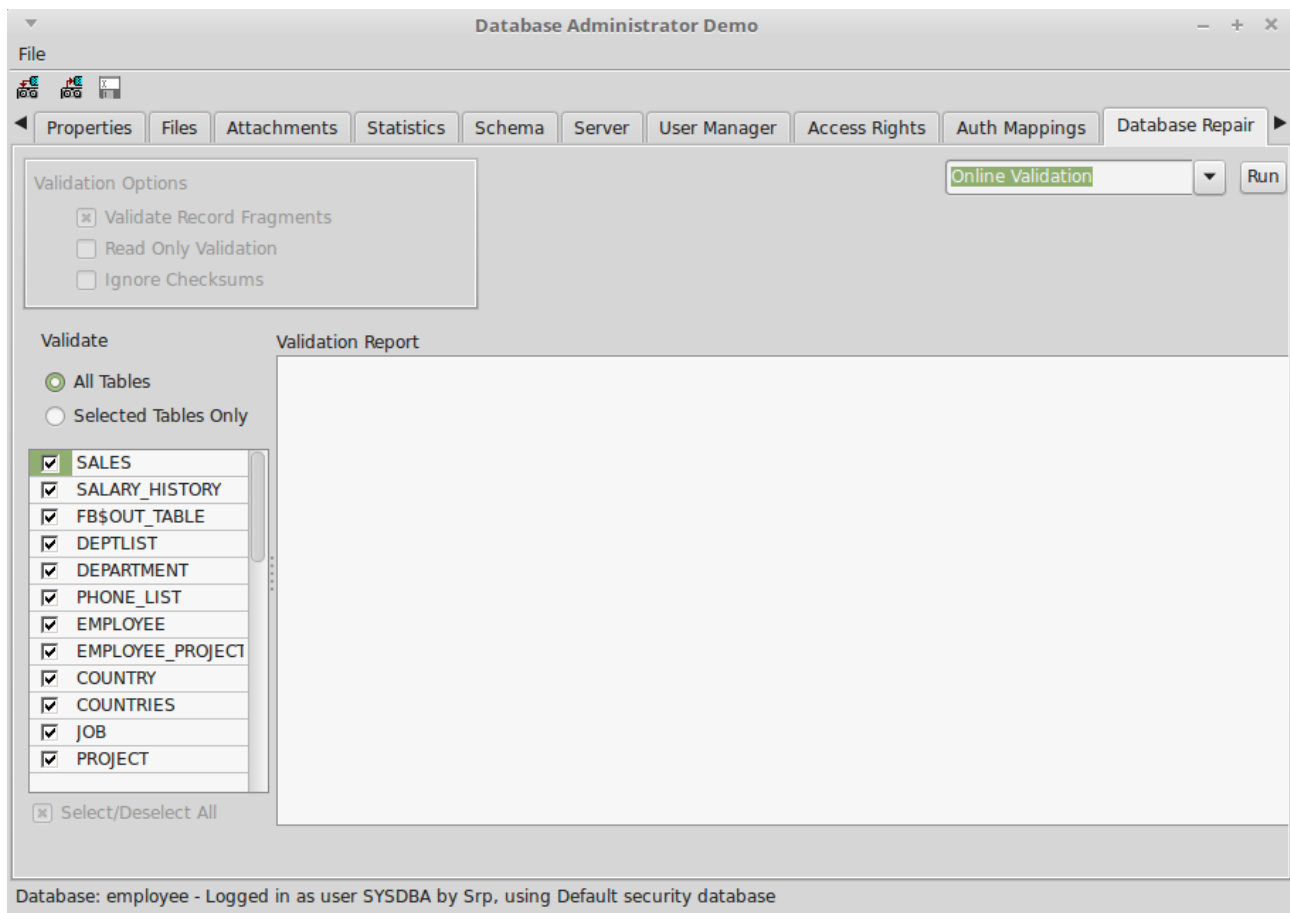


Illustration 19: Online Validation Example

3.11.3 Database Validation

Database Validation (and Repair) is more comprehensive than Online Validation, and is described as follows in the *gfix* documentation [5]:

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.

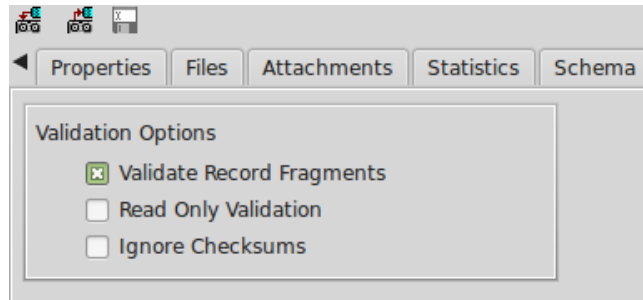


Illustration 20: The Phase One Validation Options

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).

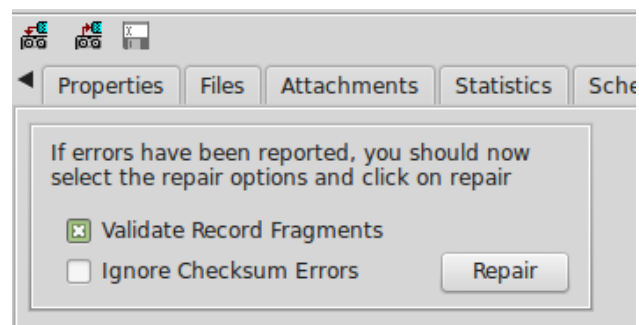


Illustration 21: Database Repair Options

3.11.4 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.

3.12 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 2.6.3.

3.13 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.

3.13.1 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.

3.13.2 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 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 2.8.6).