# Connect to an existing Database

After starting IBExpert, you will see the Database Explorer on the left side. Before a database connection can be made, the database must be registered (please refer to Register Database).

A database connection can be made to a registered database simply by double-clicking on the database alias name, displayed in the DB Explorer. There are also a number of menu options: either using the IBExpert menu item Database / Connect to Database, or the following icon:

in the Database toolbar. Alternatively the Database Explorer right-click menu may be used, or the key combination [Shift + Ctrl + C].

To automatically connect to a database when starting IBExpert use the following menu: Database Registration Info / Additional and check: *Open database when IBExpert starts*.

If you wish to connect to a registered database using a user name and password other than the user specified in the Database Registration Info, hold the [Ctrl] key down whilst using the Database Explorer double-click option, right-click context-sensitive menu or icon, to open the Database login input window.

Should there be any problems connecting to the database, use the "IBExpert Services menu item, Communication Diagnostics.

An example connecting to a remote database using the IBExpert Database menu item, Database Registration Info:

**Server = Remote**
**Server Name = <network name of the server or its ip address> e.g. OUR_SERVER Protocol = TCP/IP**
**DB File Name = <path to the database file on the server PC> e.g. “D:\DataMyDB.fdb”**

Of course Firebird/InterBase® should be installed properly on the server PC (where your database is placed) and the Firebird/InterBase® client (fbclient.dll or gds32.dll) on your local PC.

For those preferring to use SQL, the syntax is as follows:

```
CONNECT [TO] {ALL | DEFAULT} <config_opts>
| <db_specs> <config_opts> [, <db_specs> <config_opts>...];

<db_specs> = dbhandle
| {'filespec' | :variable} AS dbhandle

<config_opts> = [USER {'username' | :variable}]
[PASSWORD {'password' | :variable}]
[ROLE {'rolename' | :variable}]
```

```
[CACHE int [BUFFERS]]
```

A subset of CONNECT options is available in isql.

```
CONNECT 'filespec' [USER 'username'][PASSWORD 'password']
[CACHE int] [ROLE 'rolename']
```

The CONNECT statement:

- Initializes database data structures.
- Determines if the database is on the originating node (a local database) or on another node (a remote database).

An error message occurs if Firebird/InterBase® cannot locate the database.

- Optionally specifies one or more of a user name, password, or role for use when attaching to the database. PC clients must always send a valid user name and password. Firebird/InterBase® recognizes only the first 8 characters of a password.

**If a Firebird/InterBase® user has ISC_USER and ISC_PASSWORD environment variables set and the user defined by those variables is not in the isc4.gdb/security.fdb/security2.fdb, the user will receive the following error when attempting to view isc4.gdb/security.fdb//security2.fdb users from the local server manager connection: *undefined user name and password*. This applies only to the local connection; the automatic connection made through Server Manager bypasses user security.**

- Attaches to the database and verifies the header page. The database file must contain a valid database, and the on-disk structure (ODS) version number of the database must be the one recognized by the installed version of Firebird/InterBase® on the server, or Firebird/InterBase® returns an error.
- Optionally establishes a database handle declared in a SET DATABASE statement.
- Specifies a cache buffer for the process attaching to a database.

In SQL programs before a database can be opened with CONNECT, it must be declared with the SET DATABASE statement. isql does not use SET DATABASE. In SQL programs while the same CONNECT statement can open more than one database, use separate statements to keep code easy to read.

When CONNECT attaches to a database, it uses the default character set (NONE), or one specified in a previous SET NAMES statement.

In SQL programs the CACHE option changes the database cache size count (the total number of available buffers) from the default. This option can be used to:

- Set a new default size for all databases listed in the CONNECT statement that do not already have a specific cache size.
- Specify a cache for a program that uses a single database.
- Change the cache for one database without changing the default for all databases used by the program.

The size of the cache persists as long as the attachment is active. If a database is already attached through a multi-client server, an increase in cache size due to a new attachment persists until all the

attachments end. A decrease in cache size does not affect databases that are already attached through a server.

A subset of CONNECT features is available in isql: database file name, USER, and PASSWORD. isql can only be connected to one database at a time. Each time CONNECT is used to attach to a database, previous attachments are disconnected.

back to top of page

### Accessing a Firebird embedded database with Win1252 (or other character set)

This tip comes from Gerhard Knapp.

In order to connect to a Firebird embedded database with WIN1252 (or other character set) using IBExpert:

1. Rename fbembed.dll to fbclient.dll (always recommendable; not just in this case!).
2. Define this fbclient.dll including drive and path in the IBExpert Database Registration.
3. Specify WIN1252 in IBExpert.
4. Copy the subdirectory intl from the Program Files directory, where fbclient.dll is installed, into the directory C:\Program Files\HK-Software\IBExpert 2.0!!

You should then have no further access problems.

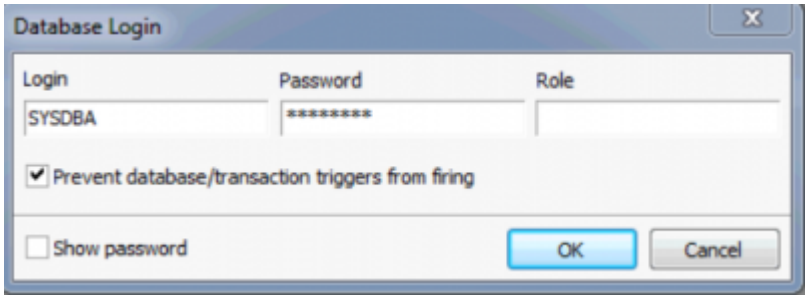
**Further information:**

When fbembed.dll is renamed fbclient.dll, it is also a fully-fledged client, i.e. if an application needs to access an embedded database on a Firebird server, the fbclient.dll is more than sufficient.

back to top of page

### Database login

If a password is not entered at the time of registering the database (see Register Database), it needs to be logged into each time the database is opened.




Specify a user name and associated password. If the user is not authorized or the password is not correct, an error message appears.

Optionally, a role may be specified. If the role has previously been GRANTed to the user name, all access privileges assigned to that role for the duration of the current session apply for that user.

The *Show password* checkbox, when activated, displays the actual password as opposed to the usual row of asterisks.

The *Prevent database/transaction triggers from firing* checkbox option is equal to the isc_dpb_no_db_triggers option in additional connection parameters.

If the user is an authorized user for that server, and if the password is correct, access is granted to the database.

back to top of page

### Remote database connect using an alias

This article was written by Claudio Valderrama ( https://www.cvalde.net/ - The InterBase® Unofficial Site), February 2002

Many developers wish to avoid the client having to give the engine the full path of the database in the same machine (node) where the engine runs? It is not only inconvenient when the database's location is changed, it is also a low level that the client shouldnt be concerned about. Finally, many developers have concerns with the security. Ideally, the physical location of the engine and the databases shouldnt be disclosed to the client. Only an Alias should be visible.

It's incredible that for years, a built-in solution in the engine (that works whenever the server is a NT machine) has been lying in the heart of the code and nobody made it public, less even documented in some help file. Perhaps because it unfortunately is a Win32 only solution, nothing that can be used on Linux, so the location of a gdb is not truly transparent.

The syntax is very simple. It has the form:

\server!share_name!database.gdb or the form

server:!share_name!database.gdb It's not a true alias, since you still know the name of the database and of course, the server machine should be known. But it helps if you need to move the database around NT servers, without having to change configuration files or recompiling programs. Here, “server” is the NetBEUI name of the NT machine, followed by the pseudo-UNC paths that IB/FB uses. Alternatively, “server” is the TCP/IP name of the NT machine, but followed by backslashes, not the typical slashes the IB's TCP syntax uses. (Really, using slashes or backslashes is not important in a typical full path, since the engine makes the adjustments, but in this case, the syntax to recognize the share demands backslashes.) The difference is that instead of a full path inside the server, a shares name in the server is used, surrounded by exclamation marks. This share points in turn to the full path of the database, so you only have to append the database's name. It has nothing do to with client-side mappings. How it works: the client library recognizes a UNC-like path and knows it's NetBEUI. Otherwise, it recognizes a TCP-like syntax thanks to the colon. Then it connects to the required server with the right network protocol and passes the remnant of the path, stripping the server's name. A routine inside the engine, named expand_share_name, will look for the backslash followed by the exclamation mark, then if a matching “!” occurs, it takes the name inside the two pairs (“!” and “!”) and will open the registry (RegOpenKeyEx) at SYSTEM\CurrentControlSet\Services\LanmanServer\Shares to extract the data

(RegQueryValueEx) in the value <share_name>, that's supposedly the name of a registered share in the server machine. It proceeds to decode the data and gets the “Path” component inside the multi-string data that's the physical path. It loads this path in its argument and returns to the caller that will continue testing to see finally if the databases name is valid and exists.

For example, given a shares name “myshare”, the registry key shown above contains a list of values that denote shares. You can find there the implicit ones such as IAS1$ (very bad, get rid of it since it points to the IIS admin dir), the NETLOGON share and “myshare”. Reading the data in the value “myshare”, the following can be seen: MaxUses=4294967295.Path=H:PROY.Permissions=127.Remark=for fb.Type=0.. The dots denote the NULL ASCII value, since this is a multi-string. The engine looks for “path” and gets the string that follows, namely H:PROY, then appends the backlash if missing. Hence, the engine uses information in the server itself to decode the full path. This path will prefix the database name when the function expand_share_name returns to the caller. An advantage is that you don't need to grant permissions on this share. You can deny anyone any right (even if NT prompts if you are sure) and you can go further: you can stop the service responsible for handling requests of NetBEUI shares. The engine reads the registry directly, so it doesn't query the network layer. It's a true hack, a commodity to avoid the inclusion of hard-coded paths in the client. If you want to change it, just change the shares information, without granting anyone any right on the share. Since the engine reads that registry location each time a connection string should be analyzed, it will get the changed name in the next attachment request. If you disabled some networks services, so that changing the share is not possible through high level interfaces, you can edit the registry directly and change the path. Beware that the each dot represents a NULL ASCII value in the example shown above, so your path should end with that value. An even nicer feature is that this works:

H:ibdevfbbuildinterbasejrd>isql \atenea!myshare!g Database: \atenea!myshare!g SQL> ^Z but it's not restricted to NetBEUI. Indeed, as noted before, you can use TCP syntax:

H:ibdevfbbuildinterbasejrd>isql localhost:!myshare!g Database: localhost:!myshare!g SQL> ^Z (Remember that there's no restriction to the name of a gdb other than the file name conventions in the platform where the engine resides. In this case, it's simply named “g”, although an extension helps the database admin.)

There are a couple of drawbacks: first, this hack is tied to Win32. (Furthermore, I don't have a way to test it on XP, but I've been informed of success with Windows 2000.) Second, when I read that internal function expand_share_name(), I found a possible buffer overrun and closed it. Revisiting the code when I wrote this article, I found a registry key handle that wasn't closed if the function gives up prematurely for lack of RAM. (I solved this second glitch in Firebird at the time I was finishing this article.)

Hence, I believe the lack of documentation comes from the untested nature of the facility.