Embedded Engine in Firebird 3

by Helen Borrie

Revised May 2018 in collaboration with Vlad Khorsun

Since Firebird 3 was released, users have sometimes been confused about the "missing" kits for Firebird embedded. We hope this article will demystify the issue and help developers migrating older embedded applications to Firebird 3 and higher.

One deployment option for Firebird is "embedded", whereby the Firebird API client connects directly to one or more databases through an instance of the database engine that is incorporated in the application's workspace.

- Before Firebird 3, this embedding was achieved on POSIX platforms by connecting the application directly through the Firebird API client library to a Classic process (Superclassic in v.2.5) without including a TCP/IP address or hostname in the path to the database file. Firebird on POSIX always had this mode.

- For Windows applications, you had to download a specially-compiled API client library (fbembed.dll) that included a single-instance Superserver engine (Superclassic in 2.5) and locate it, along with some other components, in the folder with the application executable. As with the POSIX version, the connection path would be hostless.

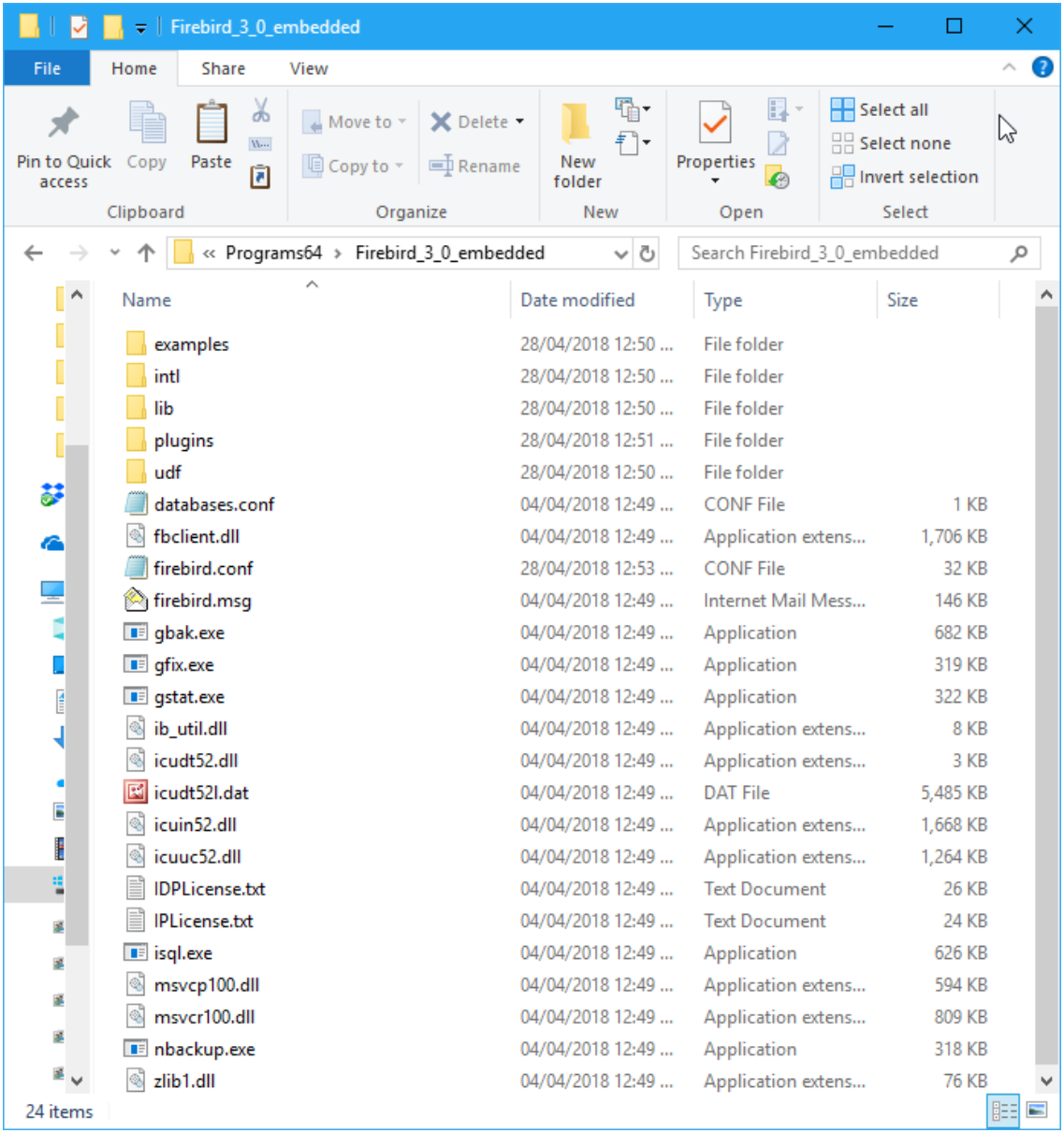
Embedded in Firebird 3

Everything you need for deploying Firebird 3 embedded with your application is present in the .zip (Windows) or .tar.gz (Linux) kit that you download from the Firebird website. We'll work with mainly with the Windows zip kit here, but the principle is the same for the Linux kit: unzip the Windows kit into its own folder or decompress and untar the Linux one into its own directory. There are differences between structures of the file systems now, specifically in the location of the binary files: Linux retains them in the `/bin` directory beneath the Firebird root, whereas Windows has them all directly in the Firebird root.

Tuning the Directory Structure

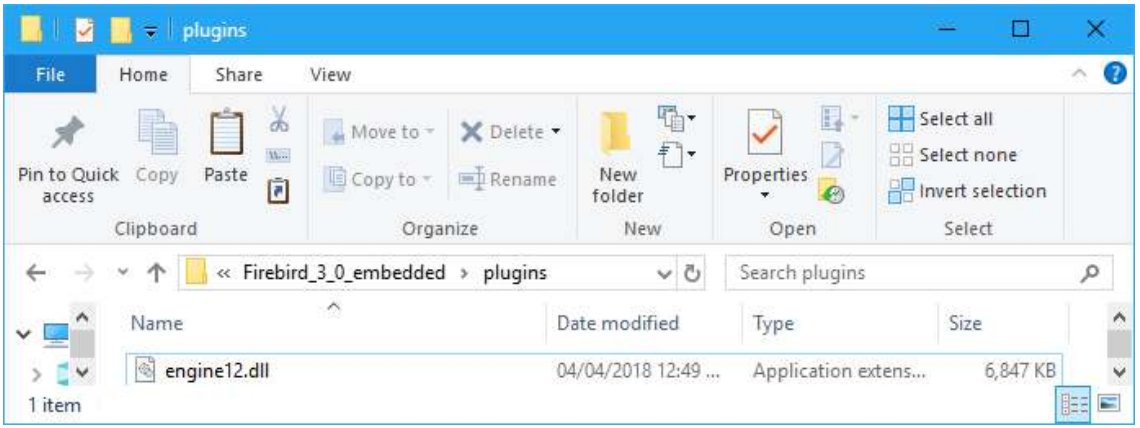
We'll make it a minimal install, on the assumption that the only executables wanted are *gfix*, *gstat*, *gbak*, *isql* and *nbackup*. (You might not want any of them.) You don't need the firebird executable at all. You don't need the security database, since authentication is not used in embedded. We are retaining the hoary old employee database in the sub-directory `/examples/empbuild` for testing, but it is not needed for deployment.

The following screenshot shows the Firebird root directory with the files and sub-directories we might want at this point:

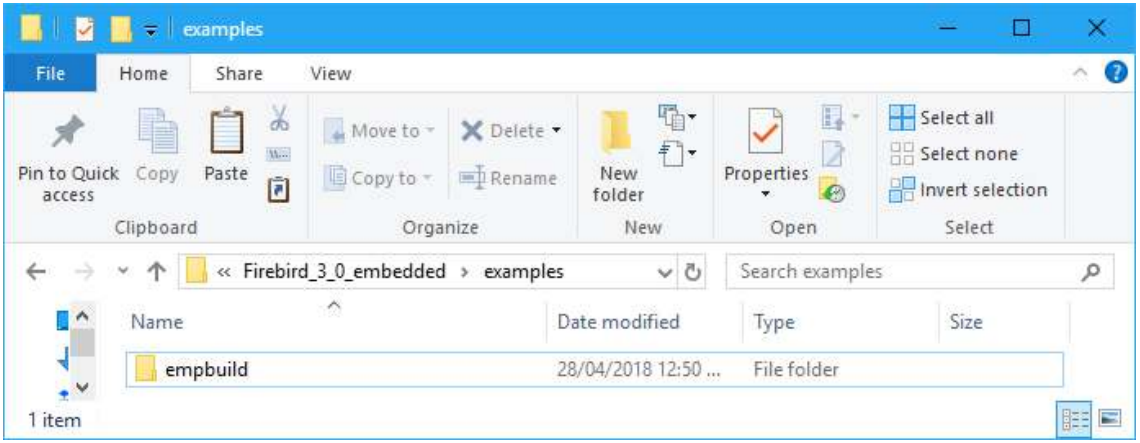


Notice that the Firebird executable—`firebird.exe` in Windows, `firebird` in Linux—is not wanted. You can take out any of the other executables that you don't intend to deploy to users, as well.

We need the `engine12.dll` (`libengine12.so`) provider module in the sub-directory `/plugins`. This is the only module we need to keep in `/plugins`, for our minimal installation. You'll want to keep others if you plan to use them for something in your application.



And just the `employee.fdb` database in `/examples/empbuild` for testing. It can be deleted once we are ready to trim down for deployment.



Configuring the Provider for Embedded

With the full unification of Firebird in Firebird 3 came plug-in providers for all of the deployment options. In the initial installation, all are enabled by default. In `firebird.conf`:

```
#Providers = Remote,Engine12,Loopback
```

The `engine12` plug-in is used for all deployment models on all platforms. `Remote` additionally enables connections from the LAN or WAN; `Loopback` additionally enables connections through the TCP/IP local loopback interface. If all are present in the configuration, the server can accept connections from any of the interfaces. Connection style is governed by the connection path.

For a simple embedded setup that mimics the pre-V.3 one, there is no need to modify the default configuration for Providers on a machine that is not running a Firebird network server. When the Dispatcher encounters a hostless connection path, it first tries the Remote provider and, as expected, fails in the absence of the networked service. Next, it tries the Engine12 provider, which should succeed if the database is not exclusively locked by a Superserver process or another embedded engine instance. The Dispatcher's last resort would be to call on the Loopback provider to attempt an XNET or INET connection, which would fail in the absence of login credentials.

If the system does have a Firebird network server running, you can eliminate it from the scope of your application by simply uncommenting the Providers parameter and deleting the `Remote` and `Loopback` options:

```
Providers = Engine12
```

Note that it is now possible for the application to pass some configuration elements via the DPB or SPB of the connection request as a string argument on the connection tag isc_dbp_config or isc_spb_config, as the case may be. It could pass "Providers = Engine12" in the attach database call, if practicable, avoiding the need to configure it specifically. The form is:

```
isc_dbp_config | isc_spb_config <string-length> "<config-fragment>"
```

E.g.,

```
isc_dbp_config 21 "Providers = Engine12"
```

When multiple parameters are configured via the isc_xxx_config tag, use the \n (newline) symbol to separate the parameters within the string, counting each \n as one character.

Bitness: Server vs Client

Because both the engine and the client are running in the same application space, you cannot mix the 64-bit client and the 32-bit engine, or vice versa. To get the benefit of the 64-bit engine, you should ensure that your application is built as 64-bit.

Illustration

To illustrate our embedded application at work, we will use *isql* to connect to the employee database, which the default installation aliases in `databases.conf` as `employee`. An

embedded connection does not authenticate, so a user name and password are not required.

```
C:\Programs64\Firebird_3_0_embedded>isql employee
Database: employee, User: HELEN
SQL> show tables;
       COUNTRY                         CUSTOMER
       DEPARTMENT                      EMPLOYEE
       EMPLOYEE_PROJECT                JOB
       PROJECT                         PROJ_DEPT_BUDGET
       SALARY_HISTORY                  SALES

SQL> exit;
```

User HELEN is just an ordinary, serverwide user on Windows, which the engine requests from the operating system if user name not passed explicitly, or obtained implicitly from the environment variables (ISC_USER, et al.).

Although no authentication is required, we still might need to log in explicitly as SYSDBA or another user with non-public privileges. In that case, we need to supply the user name. The password is not required, since authentication still does not apply. The user name will be associated with privileges and mappings in the specified database.

```
C:\Programs64\Firebird_3_0_embedded>isql employee -user sysdba
Database: employee, User: SYSDBA
SQL> exit;
```

Coexisting with a Server

Before Firebird 2.5, an embedded engine on Windows could not connect to a database that already had connections from a full server or an existing instance of embedded. The reverse is true, too: a full server could not connect to a database to which an embedded instance was connected. That was because the prior versions of the embedded database server on Windows were implemented as Superserver which, for various reasons, requires an exclusive lock on the database file. In v.2.5, embedded could share a database with another embedded engine and with a stand-alone Superclassic or Classic server on all platforms.

In the unified Firebird 3 architecture, an embedded engine is configured by default to run as a Superserver instance on both Windows and Linux. In firebird.conf:

```
#ServerMode = Super
```

As such, it needs to acquire an exclusive lock on the database file to connect and, while connected, it prevents shared connections from other engine instances. With this

configuration, it is not possible, for example, to have client/server clients connected simultaneously with browser clients attached to the same database through an intranet application that uses an embedded engine.

The solution is to run your embedded engine as a [Super]Classic process in concert with your Superclassic or Classic server. Uncomment the ServerMode parameter in the firebird.conf of your embedded structure and set it to Classic or Superclassic:

```
ServerMode = Classic
```

Note, for the embedded engine, Classic and Superclassic are equivalent.

Embedded Connections for Tools Usage

It is not necessary to construct a separate file framework if you plan to use an embedded connection to run administrative tools such as gbak or gfix or run privileged DDL in Classic or Superclassic. Simply keep the original Providers configuration and log in with a "hostless" path and whatever user name you need to accomplish the tasks. The illustrations above will work equally well from the Firebird root location.

Conclusion

That's all it takes to construct a framework for your embedded engine application in Firebird 3. You can set it up with as much or as little functionality as you want. Just add your application, stir gently, and you are good to go.

.