Getting started with Client-Server with SQLite

Whil Hentzen

"We need to get away from DBFs" is a refrain I hear regularly from fellow developers. Be it due to perceived instability of the file format, the need for tables larger than 2 GB, or the result of political machinations, the result is the same – a desire to move to a SQL database back-end. SQLite can be an excellent intermediate step – and possibly the final word - in the process of restructuring your application to talk to a SQL back-end.

You've got a FoxPro application that's been running fine for years. Lately, though, you've been feeling the need to make some changes. The overriding change in your mind is to a new back-end for the data.

You're not happy with DBFs; while they are much more robust than a decade or two ago, they are still open to corruption once in a while. More so, since they're DBFs, they're accessible from other programs - open up Excel or OpenOffice.org or even Access, and the data is there for the world to see. To say nothing of a text editor, when you think about it.

And DBFs have limits. Two gigabytes of limits, specifically. Back in the day, a two gigabyte file seemed inconceivable, but these days, not so difficult to hit. You're tired of having to archive last year's data into separate files because you're running out of room.

So you're seriously thinking about moving to a client-server architecture. You've been thinking that way for quite a while, but that move is not a trivial task.

First, there's the selection of which SQL database - SQL Server or MySQL or PostgreSQL or Oracle or.... - and that's both a technical and a financial decision. Likely some political concerns in there as well.

Next, you'll also have to learn to manage the infrastructure of a SQL database back-end. Then the installation on a dedicated server box, and setting up access from the outside - be that elsewhere on a protected network or from the Internet at large. Configuration, setting the parameters and tuning the database for the type of application you're running, and learning to use the configuration management console.

But a database isn't any good until people and applications can get at it, so you'll have to set up security - permissions and users and roles and perhaps other layers. That's another user interface to learn and master, perhaps an entire separate application.

All of this has to be done perfectly, else you'll run into a problem later on where an operation isn't work, and you tear your hair out, wondering why the code isn't working in THIS specific case, and only much later do you find out that your code is working fine, thank you very much. The problem has to do with a change in the IP connection.

Once you're comfortable with management and administration of the SQL database server, you can get back to where you wanted to be in the first place - programming. Remember programming? And the To Do list here isn't trivial either.

You may have to learn a new language - not everyone has made the leap from the xBase paradigm using SKIP and DO WHILE loops to SELECT, INSERT and UPDATE SQL commands.

It's not just lingo, though. You likely have to restructure most if not all of your data handling, away from the record-based logic that uses the aforementioned SKIP and DO WHILE commands, and to the set-based logic of SQL. Many, many applications were written with their data manipulation code intertwined with the user interface, perhaps dating back to the FoxPro/DOS and FoxPro for Windows days, like so:

```
@1,1 SAY customer->name
skip
@2,1 SAY customer->name
skip
```

to display an abbreviated list of customers. The classic VCR-style Next-Previous-First-Last navigation style buttons rely on being able to move through a table's records sequentially. Even more recent VFP applications often have bits and pieces of DBF navigation and manipulation strewn throughout their methods.

And inertia is a powerful thing, after all. It's easy to let things get in the way: "We'll tackle this as soon as <fill in the name of some calendar-based event that doesn't seem too distant, yet far enough away that planning doesn't need to be undertaken quite yet.>"

The big win

What if you didn't have to worry about the first half of this exercise quite yet? What if you could get to the Visual FoxPro programming part of this conversion in about ten minutes? What if you didn't have to worry about the SQL database selection, installation, configuration, or security setup at all?

Yes, it can be done.

SQLite (sqlite.org) is a fast, highly-reliable, zero-installation, zero-configuration, completely free SQL database that is perfect for learning to transition your application that relies on an xBase data paradigm to a client-server architecture.

But I've never heard of SQLite

Of course you haven't. Nobody has. But nobody has heard of VFP anymore, either. That doesn't mean it isn't still be used all over the world.

SQLite's claim to fame is a triple threat of having a small footprint (the EXE is less than half a megabyte), fast (I've done VFP queries against a 20 million row SQLite table that returned results before my hand was back on the mouse from the keyboard) and reliable (the automated testing done on each new release is mind-boggling.) And being public domain, it's guaranteed to be free of license fees forever.

If you search your local PC for “sqlite3”, you'll likely find a half-dozen instances where it's bundled with this piece of software or that. For example, Mozilla Firefox and Thunderbird, Google Chrome, Adobe Reader and Photoshop, Skype, Dropbox and McAfee all use SQLite as the internal file format.

It's bundled with PHP, Python and REALbasic, and is also used in dedicated devices, such as the iTouch and iPod, the Symbian OS, and the flight control software for the Airbus A350.

Using it, you'll be able to focus on learning client-server techniques without having to worry whether problems you run into are your own programming fault, or if they're due to a configuration or permissions issue with the server. More than one developer has spent embarrassing amounts of time wondering why the result set came back empty, only to find out that the server was in fact not running. (If the refrain from The IT Crowd, “Have you tried turning it off and back on again?” is going through your head, you're not alone.)

SQLite is, however, a single-user server, in that it locks the entire table when writing to it. As a result it is appropriate for certain types of desktop applications and completely inappropriate for others. A surprising number of Fox applications are single-user, for example. And others may have multiple users who primarily do queries, but writes are performed either by one user, or in extremely limited quantities by multiple users. It's also appropriate for Web-based applications that have a single 'user'.

This series of articles comprise a tutorial on what's needed to set up SQLite for use with VFP, what you need to know in order to work with SQLite independently of VFP, how to connect to SQLite from within VFP, and then introduces some unique characteristics of SQLite that you'll want to be aware of when using it.

Setting up SQLite

If you've done any investigation with other SQL databases, you know that there's a fair amount of work involved with installation, configuration, and security setup. With SQLite, only one step is required: install the SQLite ODBC driver.

Yes, that is the ONLY step needed.

SQLite consists of one 500K executable that has no installation requirements, no configuration, no permissions or user setup. The ODBC driver for SQLite is packaged with the SQLite database executable, so installing the driver automatically installs EXE as well.

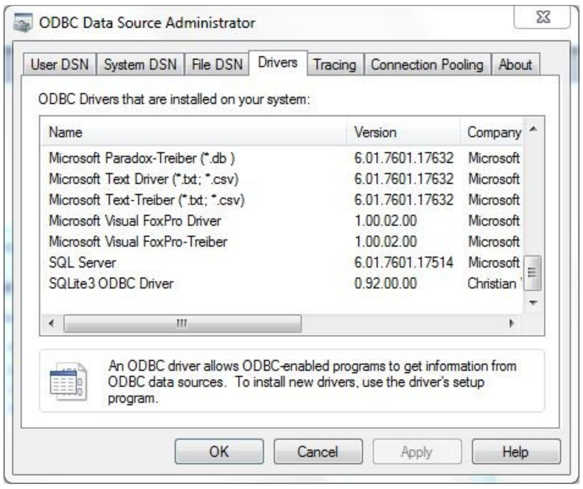
The 3.4 MB ODBC driver package (sqliteodbc.exe) is available from

```
http://www.ch-werner.de/sqliteodbc/
```

and is included with the source code for this article.

Run the EXE and it'll run you through the usual five screens to install the ODBC driver. I suggest you uncheck the SQLite2 and TCC components in the fourth screen, as I've had bad luck with the installation when they're checked, and they're not needed for our purposes.

Once you're done with the installation, you'll see the SQLite driver listed in the ODBC Data Source Administrator, as shown in **Figure 1**.


ODBC Data Source Administrator

User DSN | System DSN | File DSN | Drivers | Tracing | Connection Pooling | About

ODBC Drivers that are installed on your system:

Name	Version	Company
Microsoft Paradox-Treiber (*.db)	6.01.7601.17632	Microsoft
Microsoft Text Driver (*.txt; *.csv)	6.01.7601.17632	Microsoft
Microsoft Text-Treiber (*.txt; *.csv)	6.01.7601.17632	Microsoft
Microsoft Visual FoxPro Driver	1.00.02.00	Microsoft
Microsoft Visual FoxPro-Treiber	1.00.02.00	Microsoft
SQL Server	6.01.7601.17514	Microsoft
SQLite3 ODBC Driver	0.92.00.00	Christian

An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

Figure 1. The Drivers tab of the Data Source Administrator shows the SQLite ODBC drivers.

Connecting to SQLite

If you've gone through the process of connecting to a different ODBC data source in the past may be expecting that the next step is to set up a specific SQLite DSN for use with VFP. Supposing you did so, creating a DSN named 'slvfp'. Doing so means you could then write code like so:

```
? sqlconnect("slvfp")
```

to connect to the database.

Not so.

Unlike other SQL connectors, a DSN with SQLite must include the name of the database file - and the fully qualified path to that database if it's not in the path. That lessens the flexibility of such a connector. So, instead, we'll build a connection string that includes the location of the SQL database file that we want to use. The string would look like the string in **Listing 1**.

Listing 1. A SQLite connection string.

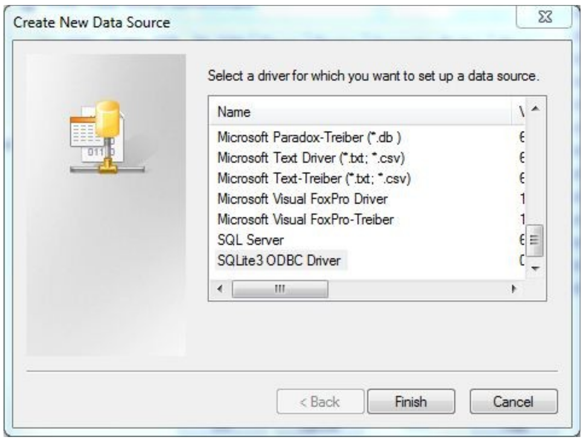
```
"DRIVER={SQLite3 ODBC
Driver};Database=f:\DatabaseOne;StepAPI=0;Sync
Pragma=NORMAL;NoTXN=1;Timeout=500000;ShortName
s=0;LongNames=1;NoCreat=0;NoWCHAR=0;FKSupport=
1;JournalMode=;OEMCP=0;LoadExt=FTS4,SEE;BigInt
=0;"
```

and can be passed to sqlstringconnect() as an argument. This may be a mouthful for the first time user; all full of syntax and possibilities for typos.

The magic question has always been, "How do you determine what the syntax of a connection string is?" Here's how, in two basic steps.

First, create a system DSN for a specific SQL database. Open the ODBC Data Source Administrator via Control Panel.

From the System DSN tab in the ODBC Data Source Administrator dialog, click the Add button, and select the SQLite driver from the alphabetically organized list, as shown in **Figure 2**.

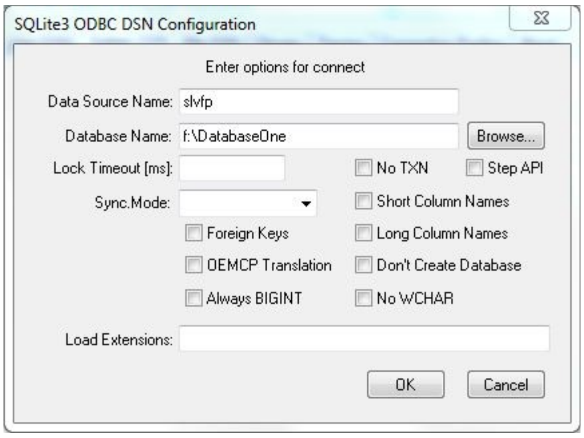

Create New Data Source

Select a driver for which you want to set up a data source.

Name
Microsoft Paradox-Treiber (*.db)
Microsoft Text Driver (*.txt; *.csv)
Microsoft Text-Treiber (*.txt; *.csv)
Microsoft Visual FoxPro Driver
Microsoft Visual FoxPro-Treiber
SQL Server
SQLite3 ODBC Driver

Figure 2. Selecting the ODBC driver for SQLite when creating the DSN.

Next, the DSN Configuration dialog appears. Fill the values for a name and point to the database name, as shown in **Figure 3**.


SQLite3 ODBC DSN Configuration

Enter options for connect

Data Source Name: slvfp

Database Name: f:\DatabaseOne

Lock Timeout [ms]:

Sync.Mode:

☐ No TXN ☐ Step API

☐ Short Column Names

☐ Foreign Keys ☐ Long Column Names

☐ OEMCP Translation ☐ Don't Create Database

☐ Always BIGINT ☐ No WCHAR

Load Extensions:

Figure 3. The ODBC Connector dialog with appropriate values filled in.

Click OK for your new DSN. End of step 1.

You may be wondering about this “DatabaseOne” file in the root of drive F. What if you don't have a “DatabaseOne” SQLite database file? This is the file that will contain everything having to do with the database – tables, indexes, and metadata. It doesn't have to already exist when you create your DSN – you can enter the name of the SQLite database you want to create from VFP if you like. (You just have to make sure you enter a valid path.) If you jump over to drive F after creating the DSN, you'll see that it doesn't exist yet. It gets created when you actually make the connection – which we'll do next.

Step 2 is extracting the connection string. Open up VFP and enter the following commands:

```
m.liHandle = sqlconnect(“slvfp”)
m.lcX = sqlgetprop(m.liHandle, ;
   “ConnectString”)
? strtofile(m.lcX, “\teststring.txt”)
```

Let's take a look at this code.

The first line connects to the SQLite database using the new DSN. The return value is either a postitive integer (the number of the connection, or 'handle' that has been established) or a negative value (indicating failure.) The handle is the value you'll use to perform all future operations, so it's a good idea to store it to a value immediately.

The second line queries the DSN's properties and places the value of the "ConnectString" property into a variable. If you type

```
=sqlgetprop(
```

into the command window, you'll see that Intellisense pops open a combo box with a dozen available DSN properties to examine.

The third command saves the value of the ConnectString property to a file that you can experiment with. In the case of SQLite, it will look like **Listing 2**:

Listing 2: A default SQLite connection string.

```
"DRIVER={SQLite3 ODBC
Driver};Database=f:\DatabaseOne;StepAPI=0;Sync
Pragma=NORMAL;NoTXN=0;Timeout=100000;ShortName
s=0;LongNames=0;NoCreat=0;NoWCHAR=0;FKSupport=
0;JournalMode=;OEMCP=0;LoadExt=;BigInt=0;"
```

This can appear to be pretty intimidating, so let's break it up into easily digestible pieces. A connection string is made up of two or more key-value pairs, each separated by a semi-colon. For readability's sake, I put each pair on its own line, as shown in **Listing 3**.

Listing 3: A typical SQLite connection string, broken into key-value pairs.

```
"DRIVER={SQLite3 ODBC Driver};
Database=f:\DatabaseOne;
StepAPI=0;
SyncPragma=NORMAL;
NoTXN=0;
Timeout=100000;
ShortNames=0;
LongNames=0;
NoCreat=0;
NoWCHAR=0;
FKSupport=0;
JournalMode=;
OEMCP=0;
LoadExt=;
BigInt=0;"
```

Each key-value pair specifies a single setting in the connection string.

A connection string for SQLite must have at least two key-value pairs. The first identifies the driver. When a new ODBC driver is released, part of the string will change, perhaps from “SQLite3” to “SQLite4”. You can determine the string from the description of the driver in Figure 2.

The second key-value pair that is required is the identification of the database. While you can get away without including the fully qualified path name here, doing do will require that the database file be discoverable via the VFP path, and that is a route fraught with danger. Better to explicitly define the path and leave no question about where the database is, or (in the case of multiple database files residing on disk, say, for test and live use cases) which version is being accessed.

What about the rest of the key-value pairs? If you examine Figure 3 carefully, you'll see that each control in the Configuration dialog maps to key-value pair in Listing 3. Key-value pairs are automatically created for each setting, with the value being set to the default value. The sudden appearance of more and more parameters in the connection string can soon lead one to think that you’ll never learn all of the possible values, and that you better just stick to the DSN wizard. Not necessarily! You can leave out the parameters that are SQLite's default values, and just explicitly define the ones that are different. If you wanted a different timeout value, for example, you'd include a key-value pair like this:

```
Timeout=5000
```

Doing so, our connection string now looks like **Listing 4**.

Listing 4. A stripped down connection string.

```
"DRIVER={SQLite3 ODBC Driver};
Database=f:\DatabaseOne;Timeout=5000;"
```

You can pass the stripped down connection string to the SQLStringConnect() function, like so:

```
m.lcXN = "{DRIVER=SQLite4 ODBC Driver};" ;
   + "Database=f:\DatabaseOne;Timeout=5000;"
liHandle = sqlstringconnect(lcXN)
```

As you can see, now that we're building the connection string in code, the location of the database can be built on the fly, instead of having to have a hard-coded path in the DSN.

Next issue, we'll discuss what can go wrong when connecting, how to recover, and start manipulating databases.

Author Profile

Whil Hentzen is a independent software developer based in Milwaukee, Wisconsin (as opposed to Milwaukee, Minnesota, as many people think.) His writing has killed many trees over the years, but none since 2007. He has realized he really sort of misses it. You can reach him at whil@whilhentzen.com