Using Actian PSQL as a Data Store with VMware vFabric SQLFire

Actian PSQL White Paper

May 2013

Contents

Introduction

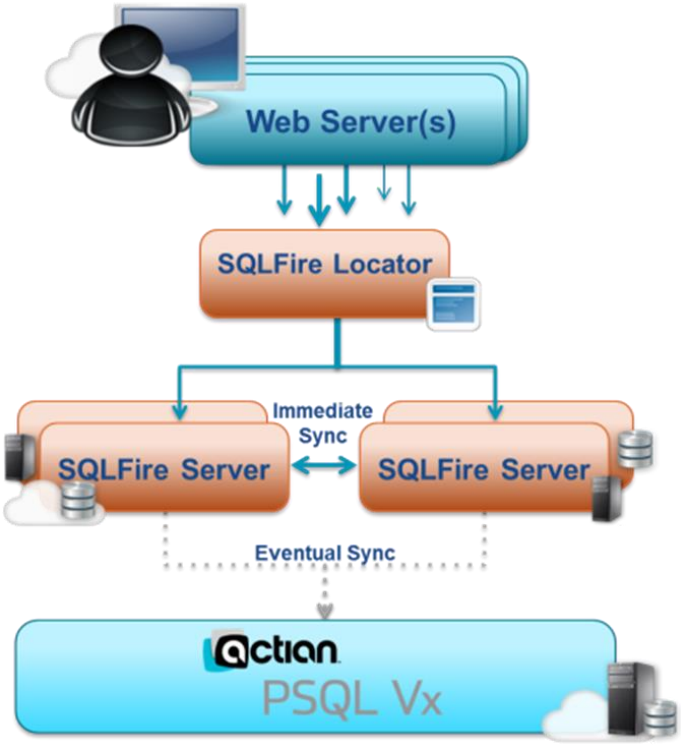
This paper is intended for application developers who are considering using VMware vFabric SQLFire as a front-end data management layer for a PSQL application. (For simplicity, this paper hereafter refers to vFabric SQLFire as just "SQLFire.") Such a layer provides the benefits of SQLFire—such as dynamic horizontal scaling with shared memory pools and scaling across geographic distances—while retaining the benefits of PSQL. In fact, SQLFire and PSQL Vx Server are well suited for each other. Both help ensure high transaction rates, high availability, are memory oriented, and optimized for cloud computing. Both support established standards such as SQL, JDBC, and ADO.NET.

This paper allows an application developer to demonstrate that data changes to the in-memory database in SQLFire are also propagated to the underlying data store, PSQL Vx Server. Even when the changes involve SQLFire failover or horizontal scaling, the database copies in SQLFire and the underlying database in PSQL all remain consistent.

In broad terms, only three pieces are required to demonstrate the use of PSQL with SQLFire:

- A SQLFire configuration using a Locator and at least two Servers
- Some data to manipulate
- PSQL Vx Server to serve as the data store

Figure 1. Components Required for the Demonstration



To help you with the demonstration, we have provided an archive file that you can download from the Actian PSQL website. Among other items, the archive file contains the schema and data from the

DEMODATA sample database. You can easily import the schema and data into your SQLFire configuration to facilitate running the demonstration.

Obviously absent from the arrangement shown in Figure 1 is an application. A typical real-world scenario would include an application that communicates with the database through the SQLFire layer. The demonstration archive file also provides the source code for a sample ADO.NET application. The sample application is discussed later in this paper. For proof-of-concept, an application is not required because the demonstration uses the SQLFire command line utility to issue commands.

Prerequisites and Assumptions

The demonstration requires certain prerequisites and assumes the following.

- You have downloaded the archive file for this demonstration, psql_sqlfire_demo.zip, from http://login.pervasive.com/download/index/16925 and extracted its contents.
- Your environment has a SQLFire configuration of three machines, either physical or virtual. (The use of three machines is not a technical requirement, but it makes the discussion easier to follow.) One machine functions as the SQLFire Locator. Each of the other two machines functions as a SQLFire Server. The Locator and the Server instances are authorized and functional on a network. SQLFire is installed on all machines in the directory c:\sqlfire.
    - Apache Ant is installed and functional on the SQLFire machine that you intend to use to import the schema and data from the DEMODATA sample database.
    - Java JDK v1.6 or later, ideally the latest build, is installed and functional on all SQLFire machines.
    - A PSQL Client is installed on the same machines as each SQLFire Server.
    - The CLASSPATH environment variable on all three machines must include the PSQL files pvjdbc2.jar, pvjdbc2x.jar, and jpscs.jar, which are required for the PSQL JDBC Client. By default, these files are installed in the *install_directory*\bin folder under Program Files.
- You have installed PSQL Vx Server on the same machine as the SQLFire Locator. PSQL Vx Server is authorized and functional on a network. The database engine is running. (The one-day temporary key is sufficient for this demonstration.)

  The PSQL Vx Server installation requires no special configuration changes to use the database engine with SQLFire. The server running the database engine can reside wherever you want in your environment, for example, on a physical machine instead of on a VM. If you already have reporting methods established for the data on that physical machine, for instance, those methods can remain unchanged. For convenience of discussion, this paper assumes that PSQL Vx Server is installed on the same machine as the SQLFire Locator.
- All of the machines run a modern version of the Windows operating system, such as Windows 2008 R2 for the Server machines and Locator machine, or Windows 7 on any client machines. All machines have the operating system installed, activated and functional on a network. (Both PSQL and SQLFire support Linux, but this demonstration uses Windows because of the ADO.NET sample application.)

Disclaimer

The content in this white paper was verified at the time of publication. However, VMware Inc. could revise SQLFire at any time, in which case portions of this paper may no longer apply.

Demonstration Steps

With the prerequisites met, you have only 5 steps required for the basic demonstration.

1. Start the SQLFire Locator.
2. Start the SQLFire Servers.
3. Import DEMODATA schema and data into one of the SQLFire Servers.
4. Start the SQLFire asynchronous event handlers on each SQLFire Server instance.
5. Propagate data changes from the SQLFire databases to the underlying data store, PSQL.

Optionally, after completing the five steps, you can review the sample application provided with the demonstration. The sample application helps you to understand the changes required to port an existing ADO.NET application for use with SQLFire. See the section below

Porting a PSQL Application to SQLFire.

Note:

When any of the following steps refer you to "start an interactive sqlf instance," change directory to the SQLFire installation directory and execute the command to start the sqlf command line utility:

```
cd /d c:\sqlfire
bin\sqlf
```

Some of the following steps may not work if you start an interactive sqlf instance from a different directory than the SQLFire installation directory.

1. Start the SQLFire Locator

Start the SQLFire Locator first, then the SQLFire Servers. The SQLFire Servers and your application communicate with each other via the Locator so you must start the Locator first. For ease of discussion, the following steps refer to the machine on which the SQLFire Locator is installed as *SQLFireDemoLoc*. Substitute the name of your machine where appropriate.

1. Open a command line interface on *SQLFireDemoLoc* and execute the following commands.

   ```
   cd /d c:\sqlfire
   mkdir locator1
   ```

   The make directory command is needed to provide a location for the SQLFire runtime files, such as logs.

2. Execute the following SQLFire (sqlf) command.

   ```
   bin\sqlf locator start -peer-discovery-port=3241 -peer-discovery-
   address=sqlfiredemoloc -dir=locator1 -client-bind-
   address=sqlfiredemoloc -client-port=1527
   ```

If desired, refer to the SQLFire documentation for more information about the SQLFire Locator.

2. Start the SQLFire Servers

For ease of discussion, the following steps refer to the machines on which the SQLFire Servers are installed as *SQLFireDemoSvr**1*** and *SQLFireDemoSvr**2***. Substitute the names of your machines where appropriate.

1. Open a command line interface on *SQLFireDemoSvr**1*** and execute the following commands.

   ```
   cd /d c:\sqlfire
   mkdir server1
   ```

2. Execute the following sqlf command.

   ```
   bin\sqlf server start -server-groups=SFDEMOSG -
   locators=sqlfiredemoloc:3241 -client-bind-address=sqlfiredemosvr1 -
   dir=./server1
   ```

3. Open a command line interface on *SQLFireDemoSvr**2*** and execute the following commands.

```
cd /d c:\sqlfire
mkdir server2
```

4. Execute the following sqlf command.

```
bin\sqlf server start -server-groups=SFDEMOSG -
locators=sqlfiredemoloc:3241 -client-bind-address=sqlfiredemosvr2 -
dir=./server2
```

If desired, refer to the SQLFire documentation for more information about SQLFire Server.

3. Import DEMODATA Schema and Data into SQLFire

You need to import the DEMODATA schema and data only once into any SQLFire Server instance within a server group. All other SQLFire Servers automatically receive the same schema and data through the SQLFire replication and synchronization logic. For this demonstration, you import the schema and data into *SQLFireDemoSvr1*. Note that *SQLFireDemoSvr2* automatically receives the same schema and data.

In the following steps, substitute the names of your machines where appropriate.

1. Locate the files **db-schema1.sql**, **db-schema1.xml**, and **data.xml** in the directory structure where you extracted the demonstration archive file. Copy the files to the **c:\sqlfire\lib\ddlutils\example** directory on *SQLFireDemoSvr**1***.

2. Start an interactive sqlf instance on *SQLFireDemoSvr1*.

3. Connect to the SQLFire Server.

   ```
   connect 'jdbc:sqlfire://sqlfiredemosvr1:1527';
   ```

   Ensure that the SQLFire command ends with a semicolon. Use the machine name with which you started the SQLFire Server (do not use “localhost”).

4. Create the DEMODATA tables.

   ```
   run 'lib\ddlutils\example\db-schema1.sql';
   ```

   Ensure that the SQLFire command ends with a semicolon.

5. Open a command prompt separate from the interactive sqlf instance and change directory to where the data.xml file resides, which is c:\sqlfire\lib\ddlutils\examples in this demonstration.

6. Open **build.xml** with a text editor and change the default host name to *SQLFireDemoSvr**1***.

   A. Within build.xml, locate the section with the “ImportDataToDB” target.

   B. Within that section, locate the “database url” tag.

   C. Within that tag, change the host name to *SQLFireDemoSvr**1***. For example, in the highlighted line below, change “localhost” to “*SQLFireDemoSvr**1**.*”

```
<target name="ImportDataToDB" description="Import the data ..">
  <taskdef classname="org.apache.ddlutils.task.DdlToDatabaseTask"
          name="ddlToDatabase"
          classpathref="runtime-classpath"/>
   <ddlToDatabase usedelimitedsqlidentifiers="false">
    <database url="jdbc:sqlfire://localhost:1527"
              driverClassName="com.vmware.sqlfire.jdbc.ClientDriver"
              username="app"
              password="app"/>
   <fileset dir=".">
     <include name="db-schema1.xml"/>
   </fileset>
   <writedatatodatabase datafile="data.xml"
                        usebatchmode="true"
                        batchsize="1000"/>
</ddlToDatabase>
</target>
```

7. Execute the following ANT command to load the data from data.xml into the SQLFire Server.

   ```
   ant -lib C:\SQLFire\lib\ddlutils\dist -lib C:\SQLFire\lib\ddlutils\lib
   -lib C:\SQLFire\lib ImportDataToDB
   ```

8. Leave the sqlf command session active because you need it for the next set of steps.

4. Start the Asynchronous Event Handlers

The asynchronous event handlers write the changes from the copy of data on the SQLFire Servers to the PSQL database. Each SQLFire Server must have the PSQL Client installed in order to write changes to the database.

In the following steps, substitute the names of your machines where appropriate. Ensure that the commands end with a semicolon.

1. At the sqlf command session on *SQLFireDemoSvr1,* execute the following SQLFire command to register the event handler. Enter the command as one line.

   ```
   CREATE ASYNCEVENTLISTENER DemoDataListener (LISTENERCLASS
   'com.vmware.sqlfire.callbacks.DBSynchronizer' INITPARAMS
   'com.pervasive.jdbc.v2.Driver,jdbc:pervasive://
   sqlfiredemoloc:1583/Demodata, Administrator, adminpassword') SERVER
   GROUPS (SFDEMOSG);
   ```

Note: Also change the user name and password in the command to match those required for the machine running PSQL Vx Server. The user account must have permissions to read, write, update and delete the PSQL Vx Server database.

2. Start the event handler on *SQLFireDemoSvr**1***.

```
CALL SYS.START_ASYNC_EVENT_LISTENER('DemoDataListener');
```

3. On *SQLFireDemoSvr**2***, start an interactive sqlf instance.

4. Connect to the SQLFire Server:

```
connect 'jdbc:sqlfire://sqlfiredemosvr2:1527';
```

5. At the sqlf command session on *SQLFireDemoSvr**2**,* execute the following SQLFire command to register the event handler. Enter the command as one line.

```
CREATE ASYNCEVENTLISTENER DemoDataListener (LISTENERCLASS
'com.vmware.sqlfire.callbacks.DBSynchronizer' INITPARAMS
'com.pervasive.jdbc.v2.Driver,jdbc:pervasive://
sqlfiredemoloc:1583/Demodata, Administrator, adminpassword') SERVER
GROUPS (SFDEMOSG);
```

Note: Also change the user name and password in the command to match those required for the machine running PSQL Vx Server. The user account must have permissions to read, write, update and delete the PSQL Vx Server database.

6. Start the event handler on *SQLFireDemoSvr**2***.

```
CALL SYS.START_ASYNC_EVENT_LISTENER('DemoDataListener');
```

5. Manipulate the Data

This step includes a couple of scenarios to demonstrate that changes to data on the SQLFire Servers are also propagated to the underlying data store, PSQL Vx Server. Even when the changes involve SQLFire failover and horizontal scaling, the database copies in SQLFire and the underlying database in PSQL all remain consistent.

In each scenario, substitute the names of your machines where appropriate.

Scenario 1 Consistent Data from DEMODATA Copy To Underlying Data Store

1. Start the sqlf command line utility on *SQLFireDemoSvr**1*** (if it is not already running).

```
bin\sqlf
```

2. Connect to the SQLFire Server (if not already connected).

```
connect 'jdbc:sqlfire://sqlfiredemosvr1:1527';
```

3. Insert a row into the “Room” table in the in-memory copy of DEMODATA.

```
INSERT INTO demodata.room VALUES ('PSQL Building', 5000, 250, 'Demo
Auditorium');
```

4. On SQLFireDemoLoc, start Pervasive Control Center (PCC) and connect to the DEMODATA database running on that machine.

   Note: This paper assumes that you are familiar with using PSQL utilities, such as PCC. For details on PCC, see the chapter "Using Pervasive PSQL Control Center" in *Pervasive PSQL User's Guide*.

5. In PCC, double-click the "Room" table for DEMODATA and verify that the new record was inserted.

   Steps 3 through 5 demonstrate that changes to the copy of DEMODATA on the SQLFire Server are propagated to the DEMODATA database in the data store.

6. On *SQLFireDemoSvr1*, use the SQLFire session opened above to experiment with other data manipulation statements to the in-memory DEMODATA database. Try some DELETE and UPDATE statements to various tables. Then, on SQLFireDemoLoc, use PCC to verify the changes to the PSQL Vx Server copy of DEMODATA.

Scenario 2 Consistent Data with Failover and Horizontal Scaling

Ensure that you create the connection to the SQLFire Servers through the SQLFire Locator instead of directly to the Server. Otherwise, as you start and stop the Servers, connections between them may not re-establish. See also the vendor documentation about SQLFire Locator.

1. Start the sqlf command line utility on *SQLFireDemoSvr**1*** (if it is not already running).

   ```
   bin\sqlf
   ```

2. Connect to the SQLFire Locator (if it is not already connected).

   ```
   connect 'jdbc:sqlfire://sqlfiredemoloc:1527';
   ```

3. Insert a row into the "Room" table in DEMODATA.

   ```
   INSERT INTO demodata.room VALUES ('PSQL Building', 5001, 12, 'Office');
   ```

4. Determine to which SQLFire Server the SQLFire Locator is connected. Execute the following command to show the SQLFire Server connection.

   ```
   show connections;
   ```

5. Stop the SQLFire Server to which the SQLFire Locator is connected.

   ```
   bin\sqlf server stop -dir=servern
   ```

   Substitute the number of the connected server for the "n" in the command. The point of stopping this SQLFire Server is to force the connection to failover to the other SQLFire Server.

   Note: Ensure that you execute the command from the SQLFire installation directory *on the Server to be stopped*. (That is, do not attempt to stop one Server from the other Server.) This paper assumes the installation directory is c:\sqlfire. The command does not work if you attempt to execute it from a different directory than the SQLFire installation directory.

6. Insert a row into the "Room" table in DEMODATA.

   ```
   INSERT INTO demodata.room VALUES ('PSQL Building', 5040, 120,
   'Classroom');
   ```

7. Execute the following command to verify that the SQLFire command line interface connection moved to the other SQLFire Server.

   ```
   show connections;
   ```

8. On SQLFireDemoLoc, start PCC if it is not already running and double-click the "Room" table for DEMODATA.

9. Verify that the new records were inserted in the PSQL copy of DEMODATA.

   As failover occurs between the SQLFire Servers, the database replicas in SQLFire and the underlying database in PSQL Vx Server all remain consistent.

You can also demonstrate horizontal scaling by completing the next 4 steps, which have you restart the SQLFire Server stopped in step 5 and verify that the server responds to queries.

10. Start the SQLFire Server that you stopped in step 5.

    A. Start the sqlf command line utility on that SQLFire Server:

    ```
    c:\sqlfire\bin\sqlf
    ```

    B. Connect to that Server:

    ```
    connect 'jdbc:sqlfire://<sqlfiredemosvr1 | sqlfiredemosvr2>:1527';
    ```

    You now have two SQLFire Servers participating in the cluster as the servers scale horizontally. Notice that you connected directly to the specific instances of the SQLFire Server as opposed to connecting to the SQLFire Locator. This is to show that you can modify data on each of those instances and have the changes propagated to the underlying database in PSQL Vx Server.

11. Insert a row into the "Room" table in the in-memory copy of DEMODATA.

    ```
    INSERT INTO demodata.room VALUES ('PSQL Building', 5200, 25, 'Scaling Lab');
    ```

12. Access PCC on SQLFireDemoLoc and double-click the "Room" table for the PSQL copy of DEMODATA. Verify that the new record was inserted.

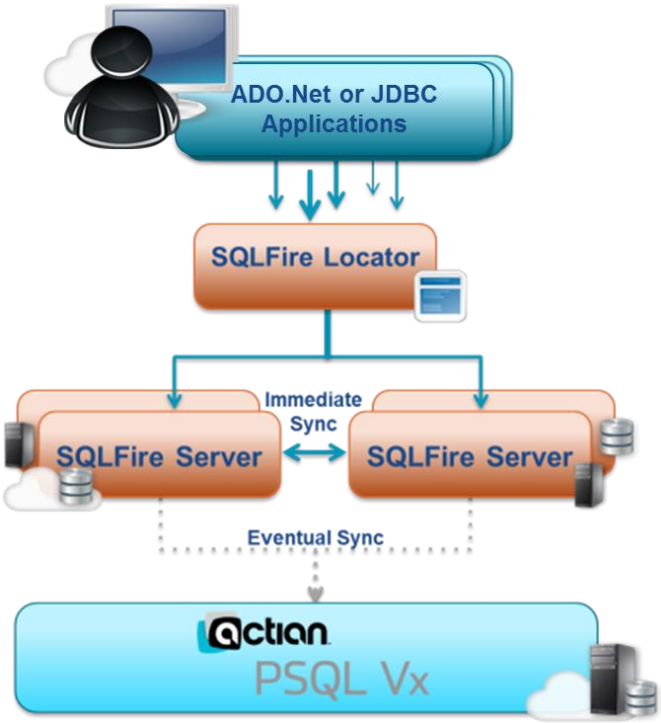
    As SQLFire Servers are scaled horizontally, the database replicas in SQLFire and the underlying database in PSQL all remain consistent.

13. Experiment with other data manipulation statements to the in-memory DEMODATA database after you stop and start SQLFire Servers. Try some DELETE and UPDATE statements to various tables. If your SQLFire configuration allows, scale several instances of SQLFire Servers and experiment with other data manipulation statements. On SQLFireDemoLoc, use PCC to verify the changes to the underlying data store in PSQL Vx Server.

Porting a PSQL Application to SQLFire

As mentioned earlier, a typical scenario includes an application that communicates with the database through the SQLFire layer. SQLFire requires that the application be ADO.NET or JDBC.

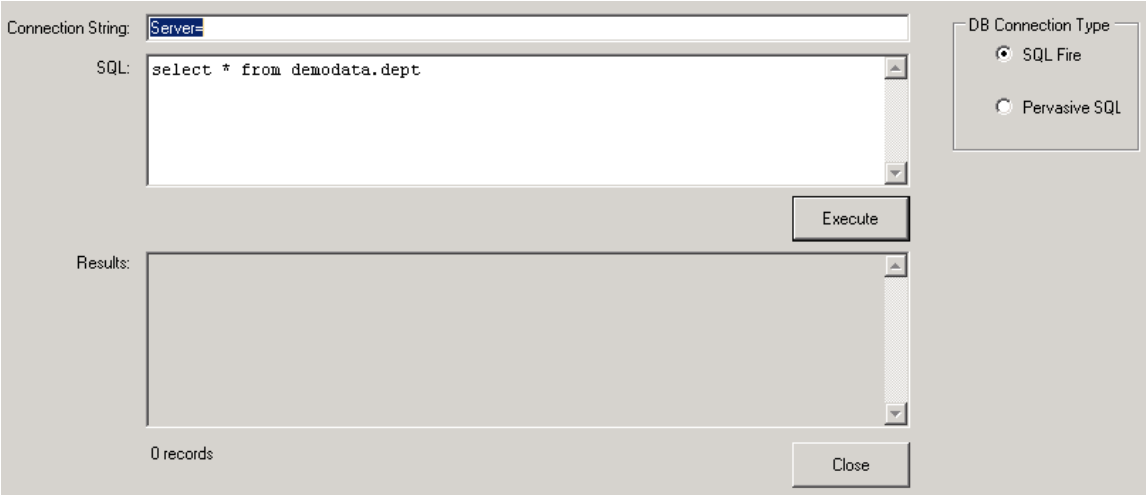
Figure 2. Typical SQLFire Configuration Using PSQL Vx Server



Typically, only minimal changes are required to port an existing PSQL application to SQLFire. If your application is already written for portability between data sources, the change simply is different connection and command objects used to connect to the database. If your application is less portable, you may also need to change the names of data types for the ADO.NET objects.

The demonstration archive file provides the source code and Visual Studio 2010 solution for a sample ADO.NET application. Figure 3 shows the GUI for the sample application.

Figure 3. ADO.NET Sample Application



Notice that the GUI allows you to select the type of database connection. The source code for the sample application shows that only two lines differ between a connection to a SQLFire Server and a connection to PSQL Vx Server.

```
If SQLFRadBut.Checked Then
   conn = New SQLFClientConnection(txtConnString.Text)
   cmd = New SQLFCommand(txtSQL.Text, conn)
Else
   conn = New PsqlConnection(txtConnString.Text)
   cmd = New PsqlCommand(txtSQL.Text, conn)
End If
```

The Sample ADO.NET Application

The source code for the sample application is available in the **ADO.NET Sample Application** directory after you extract the contents of the demonstration archive file. The main project file for the sample application is **PSQL_SQLFire_AdoApp.sln**.

You can issue data manipulation commands from the sample application. However, the primary purpose for including it with the demonstration is for you to examine the source code.

Notice that the GUI also provides a field for the string used to make a connection to the server. Only a minimal connection string is required since this is a sample application. For example, you would type the name of the server for a SQLFire connection (Server=*servername*) or the name of the DSN for a PSQL connection (ServerDSN=*dsnname*;Host=*hostname*).

You may provide additional connection string options if you choose. For details on SQLFire connection strings, refer to the SQLFire documentation. For connection strings using the PSQL ADO.NET Data Provider, see *Pervasive Data Provider for .Net Guide*.

If you want to compile and run the sample application, ensure that the following prerequisites are met.

- The machine on which you run the sample application has PSQL Client installed and has access to the SQLFire Client files. The machine is functional on a network.

  The PSQL Client is required only because the application is a client for both PSQL and SQLFire. The PSQL Client installation requires no special configuration changes to use with SQLFire.

- Visual Studio 2010 is installed and functional on a developer machine. Visual Studio is used to compile the ADO.NET sample application.

- The PSQL ADO.NET SDK is installed on the same machine as Visual Studio 2010. The SDK is available for download from www.pervasivedb.com.

Compile the sample application as you would any Visual Studio 2010 project.

1. Open the main project file, PSQL_SQLFire_AdoApp.sln, in Visual Studio 2010.
2. Adjust the paths to the PSQL ADO.NET and SQLFire references.
3. Compile the application.
4. Copy the executable to the machine on which is installed the PSQL Client and the SQLFire Client. It may be necessary for the SQLFire ADO.NET Client files to reside in the same directory as the sample application executable.

Note that the application requires access to the libraries for SQLFire and for PSQL ADO.NET.