DB/C Newsletter

August 2005

News and Comment

Thank you to all who have volunteered to help beta test DB/C DX 14.0. And thanks for your patience. We have had a few customers doing alpha testing, and based on their feedback we've added a few features and changed some others. We actually do expect to begin beta testing in September.

This month's newsletter article describes an example of a Java program that connects with DB/C FS server to do file-oriented I/O (not SQL). The example is also useful for .NET programmers because of the strong similarity between the DB/C FS Java File I/O API and the DB/C FS .NET File I/O APIs.

If you have any suggestions for other sample programs that might be useful to you, let me know and maybe we can turn your suggestion into a future newsletter.

don.wills@dbcsoftware.com

****

Sample Java Program for DB/C FS File I/O

There are two different access methods for reading and writing data with DB/C FS. The SQL access method, which comes in the forms of ODBC and JDBC, is the method that the majority of DB/C FS users are using. The lesser used DB/C FS access method is the file I/O access method. This access method requires programming to use and comes in the form of C source code, a Java library, and a .NET library (or assembly).

The example program, Demo.java, is available at the www.dbcsoftware.com web site in the Sample Code area. This is a standalone Java program that will access a file managed by the DB/C FS server. The access is via TCP/IP and the Java File I/O class library which is named fs.jar in the DB/C FS distribution. To compile the Demo.java program, you will need a Java SDK or other Java compiler environment, as well as the fs.jar file. To run the Demo.java program, you will need to have the DB/C FS server set up and running on a computer that is accessible to the computer that will be used to run the Demo.java program.

The program assumes that an empty file named vendor.txt along with three index files is available in the default directory of the DB/C FS server and that the data and index files were created by running these commands:

```
create vendor -d
index vendor 1-6
index vendor vendor2 7-26 1-6
aimdex vendor 7-26 94-133
```

Note that the vendor.txt file is a DATA type file which means that the record delimiters are linefeed characters only.

Exception handling is provided in the actual Demo.java program, but will be ignored in the following excerpts from the source code. All of the Java classes for DB/C FS file I/O are in the com.dbcswc.fs package.

The Connection class is the anchor for all operations. The first thing we can do is use the static method called getgreeting to see if the DB/C FS server is running. Here is the code to do that:

```
String server = "localhost";
System.out.println(Connection.getgreeting(server));
```

If DB/C FS is running and can be found, then a message such as "DB/C FS 4.0" is displayed.

A Connection is created to make a persistent connection to the DB/C FS server. Note that this is actually a "log on" to the server. The first parameter of the constructor is the server, the second is the database, the third is the user name, and the last is the user password. Here is the code:

```
Connection connection = new Connection(server, "demodb",
     "testuser", "testpass");
```

Records will be read and written using a buffer. In our example, the vendor record will be fixed length of 148 characters. We need to declare the record buffer like this:

```
char[] buffer = new char[148];
```

Next we declare the 3 File instances through which access will occur. The first index is an ISAM index on the vendor number field which is in positions 1-6 of the record. The second index is an ISAM index on the vendor name, vendor number fields which are positions 7-26, 1-6. The third index is an AIM index on the vendor name and vendor contact which are in positions 7-26 and 94-133. The methods to set the various access parameters are pretty straightforward. Here is that code:

```
File vendor1 = new File(connection);
vendor1.setindexfile("vendor");
vendor1.setoptions(File.DATA | File.IFILEOPEN);
vendor1.setrecsize(148);
vendor1.setrecordbuffer(buffer);

File vendor2 = new File(connection);
vendor2.setindexfile("vendor2");
vendor2.setoptions(File.DATA | File.IFILEOPEN);
vendor2.setrecsize(148);
vendor2.setrecordbuffer(buffer);

File vendor3 = new File(connection);
vendor3.setindexfile("vendor");
vendor3.setoptions(File.DATA | File.AFILEOPEN);
vendor3.setrecsize(148);
vendor3.setrecordbuffer(buffer);
```

The files are opened using this code:

```
vendor1.open();
vendor2.open();
vendor3.open();
```

Records are then written to the data file and indexed inserts are done, just like in DATABUS. Here is the code to add one record:

```
String record = "000101Best Widget Company 300 Main Street    " +
                "                      South Park     CO80440  " +
                "   Kenny McCormick                          71" +
                "9-555-1212   ";
record.getChars(0, 148, buffer, 0);
vendor1.writekey(148, "000101");
vendor2.insertkey("Best Widget Company 000101");
vendor3.insertkeys(148);
```

The first two lines of this code just move the record into the record buffer. The writekey method writes the record and the key. The insertkey and insertkeys methods insert keys into the second ISAM index and the AIM index, respectively.

The following code reads records in key sequential order using the vendor name index and displays each company name on the console:

```
vendor2.readkey("                          ");
while (true) {
     int n1 = vendor2.readkeynext();
     if (n1 < 0) break;
```

```
            System.out.println(String.valueOf(buffer, 6, 20));
        }
```

The following code uses the AIM index to retrieve all of the records that contain "mick" in the vendor contact field:

```
        int n1 = vendor3.readaim("02Fmick");
        while (n1 > 0) {
            System.out.println(String.valueOf(buffer, 6, 20));
            n1 = vendor3.readkeynext();i
        }
```

Finally, this code closes the files and disconnects from the server:

```
        vendor1.close();
        vendor2.close();
        vendor3.close();
        connection.disconnect();
```

That's all there is to it! Happy programming.

****

DB/C DX Class Schedule

Class: DB/C DX Fundamentals
Date: November, 2005
Location: Woodridge, Illinois

For information, send email to admin@dbcsoftware.com.

****

Subscribing to the DB/C Newsletter

If you don't already have the DB/C Newsletter delivered to your email address and would like to have it emailed to you monthly, just send an email message to 'dbcnews-subscribe@dbcsoftware.com'. The newsletter will be delivered to the email address from which the message was sent.