

OpenClinica: Towards Database Abstraction, Part 1

Author: Tom Hickerson, Akaza Research

Date Created: 8/26/2004 4:17 PM

Date Updated: 2/28/2005 6:01 PM,

Document Version: v0.3

Document Summary

This document was originally intended as an internal draft to be distributed to developers early on in the development phase of the OpenClinica platform. We had originally developed OpenClinica to run on top of an Oracle 10g database, and later supported a separated version running on the PostgreSQL open source database. To merge these into a common (more manageable) code-base we were looking for that magic abstraction layer that would be independent of a database, but allow us to access to the same data schema and data stored in different relational databases without a plethora of different configuration files or file structures. This first part of a two-part document is going to talk about an open source package, the Apache Commons Digester, which allows storing and importing of variables in XML very easy; so now we can store all our database-specific SQL outside of the application. In the second half, we will show source code from the new package `org.akaza.openclinica.dao.core`, which will take most of the “plumbing code” out of creating Data Access Objects in OpenClinica.

Revision History

Date	Version	Description of Document Updates	Author
8/26/2004	0.1	Creation of document, describe example code and Apache Commons Digester	thickerson
9/16/04	0.2	Addition of SqlInitServlet and description of how to use the digester in OpenClinica	thickerson

The Story So Far

OpenClinica serves as an interface for a large, protected database; with the need to create intricate data structures and permissions, also comes the need to create an interface to manage data in a seamless and comprehensive fashion. In developing a version of OpenClinica which can exist on top of either an Oracle or a PostgreSQL database (with porting to DB2 and SQL Server planned in forthcoming releases), we felt there was a need to make our J2EE code database-agnostic; in examining the open-source project known as Apache Commons Digester, most of the pain has been taken out of parsing XML and we can put SQL queries there, virtually eliminating the need to create database-specific classes.

Installing Apache Commons Digester

Most of the examples in this document are based on the article in Java World, "Simplify XML File Processing with the Jakarta Commons Digester" (see link below). To have this working on your own system, you will have to import the jars from Digester, together with some of the other commons jars, namely commons-beanutils, commons-collections-2.1.1, and commons-logging-api. As of the writing of this document, they should be part of lib.tar which is available on the OpenClinica Portal website. Note that these jars will have to be present in your Eclipse project classpath, your Ant classpath, and your Tomcat 5.0 classpath in order for you to compile, deploy and test.

Once you have these resources available, you can create xml files like the following:

```
<?xml version="1.0"?>
<queries>
  <query>
    <name>userInsertQuery</name>
    <sql>INSERT INTO USER (USER_ID, USER_NAME, USER_PASS) VALUES
(USER_SEQ.NEXTVAL,?,?)</sql>
  </query>
  <query>
    <name>userSelectQuery</name>
    <sql>SELECT USER_NAME, USER_EMAIL, PERSON_ID FROM PERSON_USER</sql>
  </query>
  <query>
    <name>userSelectQueryByProject</name>
    <sql>SELECT USER_NAME, USER_EMAIL, PERSON_ID FROM PERSON_USER WHERE
DEFAULT_PROJ=?</sql>
  </query>
</queries>
```

And then begin to parse them with code like this:

```
FileInputStream fis = new FileInputStream("../ +
File.separator +
"webapps" + File.separator +
"OpenClinicaDemo" + File.separator +
"properties" + File.separator +
"user_dao.xml");
Digester digester = new Digester();
Digester.addObjectCreate("queries/query", "Query");
digester.addCallMethod("queries/query/name", "setQueryName");
digester.addCallMethod("queries/query/sql", "setSqlStatement");
digester.parse(fis);
```

As the Java World article describes, the Digester is tricky, since only the last object popped off the stack remains after it has parsed a file. You can avoid this by pushing the object to the stack to be processed by a second method, as in the example below:

```
public void run() throws IOException, SAXException {
    Digester digester = new Digester();
    digester.push(this);

    digester.addCallMethod("queries/query", "setQuery", 2);
    digester.addCallParam("queries/query/name", 0);
    digester.addCallParam("queries/query/sql", 1);
    digester.parse("user_dao.xml");
}

//this method gets called by the digester

public void setQuery(String name, String query) {
    queries.put(name, query); //hash map that stores the query
}
```

Digester Usage in OpenClinica

Loading the digester in every command servlet would be a little performance hit, so instead of allowing that to happen, our current structure allow for us to have all the digesters set up in a servlet that runs when Tomcat starts up, loading the digesters into a static resource that can then be used later. All the developer will have to do is put the following lines in his or her code:

```
private SqlFactory factory = new SqlFactory();
...
OpenClinicatypeDAO rdao = new
OpenClinicatypeDAO(sm.getDataSource(), factory.getDigester("OpenClinicatype_dao.xml"));
```

The factory should already have the digester contained in memory; an example of the startup servlet and the static resource, SqlFactory, is included in the appendix.

By using a static resource in the Tomcat servlet container, we are able to hold the queries in the application's memory, and not have to read the file every time a user logs in or uses the database.

Conclusions

Use of the Digester helped us by not having to write an entirely new data layer to support PostgreSQL, and it may allow us to use more external files in the future and trim down code. In the Appendix, we show examples of code which has been deployed on OpenClinica, and can be accessed via CVS to review and deploy as desired. In Part 2 of this report, we describe how to abstract out a lot of code from our current system of Data Access Objects and begin to use the queries stored in XML.

Resources

Erik Swenson's Java World Article:

<http://www.javaworld.com/javaworld/jw-10-2002/jw-1025-opensourceprofile.html>

Digester Home Page:

<http://jakarta.apache.org/commons/digester/>

Appendix 1: Source Code for DaoDigester.java

```
/*
 * Created on Aug 25, 2004
 *
 */
package org.akaza.openclinica.dao.core;

import org.apache.commons.digester.*;
import java.io.*;
import java.util.HashMap;
import org.xml.sax.SAXException;

/**
 * <P>DaoDigester.java, by Tom Hickerson 8/25/2004
 *
 * <P>Creates an Apache Commons Digester, which parses SQL queries in XML and
 * then stores them in a hashmap, to be accessed later. Idea is to create one
 * XML file per Data Access Object, so that SQL syntax can be abstracted out
 * of the Java JDBC code.</P>
 *
 * @author thickerson
 *
 * TODO
 */
public class DaoDigester {

    private HashMap queries = new HashMap();
    private FileInputStream fis;

    public void run() throws IOException, SAXException {
        Digester digester = new Digester();
        digester.push(this);
        //set up a simple format for grabbing queries through XML
        /*
         * <queries>
         *     <query>
         *         <name>userDaoInsert</name>
         *         <sql>INSERT INTO USER (USER_ID, USER_NAME,
USER_PASS)
         *         VALUES (USER_ID_SEQ.NEXTVAL,?,?);</sql>
         *     </query>
         * </queries>
         */
        digester.addCallMethod("queries/query", "setQuery", 2);
        digester.addCallParam("queries/query/name", 0);
        digester.addCallParam("queries/query/sql", 1);
        digester.parse(fis);
    }

    public void setQuery(String name, String query) {
        queries.put(name, query);
    }

    public String getQuery(String name) {
        return (String)queries.get(name);
    }

    public void setInputStream(FileInputStream fis) {
        this.fis = fis;
    }
}
}
```

Appendix 2: Source code for datainfo.properties:

```
dbURL=jdbc:oracle:thin:@192.168.1.100:1521:OpenClinicadb
filePath=C:\\temp\\
dataBase=oracle
```

Appendix 3: Source code for user_dao.xml

```
<?xml version="1.0"?>
<queries>
  <query>
    <name>userInsertQuery</name>
    <sql>INSERT INTO USER (USER_ID, USER_NAME, USER_PASS) VALUES
(USER_SEQ.NEXTVAL,?,?)</sql>
  </query>
  <query>
    <name>userSelectQuery</name>
    <sql>SELECT USER_NAME, USER_EMAIL, PERSON_ID FROM PERSON_USER</sql>
  </query>
  <query>
    <name>userSelectQueryByProject</name>
    <sql>SELECT USER_NAME, USER_EMAIL, PERSON_ID FROM PERSON_USER WHERE
DEFAULT_PROJ=?</sql>
  </query>
</queries>
```

Appendix 4: Source code of SqlInitServlet.java, which loads the digesters on Tomcat startup:

```
/*
 * Created on Sep 8, 2004
 *
 *
 */
package org.akaza.openclinica.dao.core;

import javax.servlet.http.*;
import javax.servlet.*;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.*;
/**
 * <P><b>SqlInitServlet.java</b>, servlet designed to run on startup, gathers all
 * the SQL queries and stores them in memory. Runs the static object
 * SqlFactory, which reads the properties file and then processes all the
 * DAO-based XML files.
 * @author thickerson
 *
 */
public class SqlInitServlet extends HttpServlet {
    private Properties params = new Properties();

    public void init() throws ServletException {
        try {
            params.load(new FileInputStream(
                ".." + File.separator +
                "webapps" + File.separator +
                "OpenClinicaDemo" + File.separator +
                "properties" + File.separator +
                "datainfo.properties"));
        }
    }
}
```

```
        String dbName = params.getProperty("dataBase");
        SqlFactory factory = new SqlFactory();
        factory.run(dbName); //LOADS ALL QUERY FILES HERE
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
}
```

Appendix 5: Partial code from SqlFactory.java:

```
/*
 * Created on Sep 8, 2004
 *
 */
package org.akaza.openclinica.dao.core;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.*;

import org.xml.sax.SAXException;

/**
 * @author thickerson
 *
 */
public class SqlFactory {
    private static Hashtable digesters = new Hashtable();
    private String dbName = "";

    public SqlFactory() {
        //creator, do nothing here
    }

    public void addDigester(String name, DaoDigester dig) {
        digesters.put(name, dig);
    }

    public DaoDigester getDigester(String name) {
        return (DaoDigester)digesters.get(name);
    }

    public void run(String dbName) {
        //we get the type of the database and run the factory, picking
        //up all the queries. NOTE that this should only be run
        //during the init servlets' action, and then it will
        //remain in static memory. tbh 9/8/04

        ArrayList fileList = new ArrayList();
        if ("oracle".equals(dbName)) {
            fileList.add("user_dao.xml");
            fileList.add("OpenClinicatype_dao.xml");
            //currently hard-coded, but could be in a file
        } else if ("postgres".equals(dbName)) {
            //fileList.add("pg_user_dao.xml");
            //...
            //add files here as we port over to postgres, tbh
        } //should be either oracle or postgres, but what if the file is gone?
        for (Iterator fileIt = fileList.iterator(); fileIt.hasNext(); ) {
            String fileName = (String)fileIt.next();
            DaoDigester newDaoDigester = new DaoDigester();
            try {

                newDaoDigester.setInputStream(new FileInputStream("../" +
```

```
        File.separator +
        "webapps" + File.separator +
        "OpenClinicaDemo" + File.separator +
        "properties" + File.separator +
        fileName));
    try {
        newDaoDigester.run();
        digesters.put(fileName,newDaoDigester);

        } catch (SAXException saxex) {
            saxex.printStackTrace();
        } //end try block for xml
    } catch (IOException ioe) {
        ioe.printStackTrace();
    } //end try block for files
} //end for loop
}
}
```