Appendix E

Java Code used in Dry Deposition Model

Listing of Java Program Components

Main Executeable Modules

Main.java 3
TestSummarizer.java 8
TestSummarySummarizer.java 10
TestDoSurrogate 13
TestImporter.java 15

Calculation Assistance Components

Air.java 16
AverageAir.java 18
Water.java 21
AverageWater.java 22
Wind.java 24
AverageWind.java 26
Station.java 31
MyGregorianCalendar.java 36

Main Calculation I/O Components

ReaderDriver.java 39
OutputDataCreater.java 50
WriterDriver.java 67
AvgAirTempReader.java 78
AvgWaterTempReader.java 80
PropertiesViewer.java 83

Post-Calculation Summary Components

OutputFileSummarizer.java 85
SummarySummarizer.java 94
SheetImporter.java 110

Main.java –

Uses ReaderDriver, OutputDataCreater, and WriterDriver to read and process input from various sources. May be operated in single or batch mode according to a configuration file.

```
package Batch2;
import java.io.*;

/**
 * Main - Runs the main program.  This operation includes
 *  parsing any number of input files, doing summaries of
 *  the output data, generating excel spreadsheets,
 *  surrogating output data, and importing a master
 *  report.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.TestSummarySummarizer
 * @see Batch2.OutputFileSummarizer
 */
public class Main {

    /** Hint for config.properties file location */
    public static final String CONFIG_FILE = getPackageName()+".config";
    /** Batch path/file from config file */
    public static final String BATCH_FILE  = PropertiesViewer.getString(CONFIG_FILE,"BatchFileLocation")+"\\macro.txt";
    /** Output files' root path from config file */
    public static final String OUT_PATH = PropertiesViewer.getString(CONFIG_FILE,"OutputFileRootPath");
    /** Macro Mode operation from config file */
    public static final boolean MACRO_MODE = PropertiesViewer.getBoolean(CONFIG_FILE,"BatchMode");


    /** Main method-- perform dpr summaries, surrogates, etc... */
    public static void main(String[] args)
    {
        System.out.println("Config File: "+CONFIG_FILE+".properties");
        BufferedReader br = null;

          // Perform Main Program //
        if(PropertiesViewer.getBoolean(CONFIG_FILE,"DoMainProgram") )
        {


            try {
                if( MACRO_MODE )
                    br = new BufferedReader(new FileReader(BATCH_FILE)); // Connect br to <stdin>
                else
                    br = new BufferedReader(new InputStreamReader( System.in )); // Connect br to <stdin>
            } catch( Exception e )
            {
                e.printStackTrace();
                System.exit(0);
            }
            String inputFileName;
```

```
        boolean run_once = true;

        try {
            while( MACRO_MODE || run_once )
            {
                run_once = false;
                System.out.print("Input File Name:");
                if( MACRO_MODE )    inputFileName = br.readLine();
                else                inputFileName = PropertiesViewer.getString(CONFIG_FILE,"InputFileName");
                System.out.println(""+inputFileName );

                String site1,site2;
                int ii = 0;

                System.out.print("Site1:");
                if( MACRO_MODE )    site1 = br.readLine();
                else                site1 = PropertiesViewer.getString(CONFIG_FILE,"Site1");
                System.out.println(""+site1);

                System.out.print("Site2:");
                if( MACRO_MODE )    site2 = br.readLine();
                else                site2 = PropertiesViewer.getString(CONFIG_FILE,"Site2");
                System.out.println(""+site2);

                System.out.print("Enter Samples: ");
                try {
                    ii = Integer.parseInt(br.readLine());
                }   catch(IOException ee ){ System.exit(1); }
                System.out.println(""+ii);

                System.out.print("Enter Cap: ");
                try {
                    if( MACRO_MODE )    OutputDataCreater.CAP = Double.parseDouble(br.readLine());
                    // deposition cap is read from file by default
                } catch(IOException ee ){ System.exit(1); }
                System.out.println(""+OutputDataCreater.CAP );

                System.out.print("Enter diaPM: ");
                try {
                    if( MACRO_MODE )    OutputDataCreater.diaPM = Double.parseDouble(br.readLine());
                    // diaPM is read from file by default
                } catch(IOException ee ){ System.exit(1); }
                System.out.println(""+OutputDataCreater.diaPM );


                System.out.print("Enter P_fne: ");
                try {
                    if( MACRO_MODE )    OutputDataCreater.PFne = Double.parseDouble(br.readLine());
                    // diaPM is read from file by default
                } catch(IOException ee ){ System.exit(1); }
                System.out.println(""+OutputDataCreater.PFne );
```

```
                    System.out.print("Enter P_crs: ");
                    try {
                        if( MACRO_MODE )    OutputDataCreater.PCrs = Double.parseDouble(br.readLine());
                        // diaPM is read from file by default
                    } catch(IOException ee ){ System.exit(1); }
                    System.out.println(""+OutputDataCreater.PCrs );

                    System.out.print("Enter P_lrg: ");
                    try {
                        if( MACRO_MODE )    OutputDataCreater.PLrg = Double.parseDouble(br.readLine());
                        // diaPM is read from file by default
                    } catch(IOException ee ){ System.exit(1); }
                    System.out.println(""+OutputDataCreater.PLrg );




                    OutputDataCreater.calcDepositionRateTableName =
OutputDataCreater.getTableName(OutputDataCreater.diaPM);
                    String outputFileName = getOutputPath(site1,site2, OUT_PATH ); // <-- this changes

                    if( PropertiesViewer.getBoolean(CONFIG_FILE,"UseDPMHackFor2.5_10_20") )
                       if(  OutputDataCreater.diaPM == 25.0 )
                           OutputDataCreater.diaPM = 20.0;



                   System.out.println( " Input filename: "+inputFileName );
                   System.out.println( "Output filename: "+outputFileName );


               /*   Attempt to create the directory if it doesn't exist
               *     If the file already exists, quit.
               */
                   try {
                        File testCase = new File( outputFileName );
                        if( testCase.exists() )  {
                            if( MACRO_MODE )  System.out.print("File already exists at this location. Overwriting..." );
                            else {
                                System.out.print("File already exists at this location. Overwrite? [y/n]" );
                                if( br.readLine().compareToIgnoreCase("n") == 0 )  {
                                    System.out.println("Please change the path or parameters in the config file.");
                                    System.exit(0);
                                }
                            }

                        }
                        testCase.getParentFile().mkdirs();
                        System.out.println("Path OK.");
                   } catch( Exception e0 ) {
                       System.out.println("Exception: "+e0);
                       System.exit(1);
                   }
```

```
                ReaderDriver rd = null;

                rd = new ReaderDriver( inputFileName, ii, br );
                rd.parseInputFileForVariableCollection( site2 );

                WriterDriver cd;

                OutputDataCreater opd = null;
                /************print the header of the output file****************/
                try{
                    PrintWriter outputFile = new PrintWriter(new FileOutputStream(outputFileName));
                    outputFile.close();
                }catch(FileNotFoundException e){
                    System.out.println("File Not Found Exception Occured: "+e.toString());
                }

                while((opd = rd.createODCfromInput())!=null){
                    cd = new WriterDriver(opd);
                    cd.writeToFile(outputFileName);
                    cd = null;
                }
                opd = null;
                rd = null;
                System.out.println("Output Data complete!");
                WriterDriver.m_cd144Stream.close();

                OutputFileSummarizer.doSummary(outputFileName);


                try {
                    Runtime.getRuntime().exec("explorer "+convertSlashes((new File(outputFileName)).getParent()));
                } catch( Exception e ) { }
            }
            OutputDataCreater.closeConnection();
        } catch( Exception e ) {
            e.printStackTrace();
            System.out.println("Done!");

        }

    }
     TestSummarySummarizer.main(args);
}


private static String convertSlashes( String in )
{
int i;
    while( (i=in.indexOf("/")) != -1 )
        in = ""+in.substring(0,i-1)+"\\"+in.substring(i+1);
    return in;
}
```

```
    private static String getOutputPath( String site1, String site2, String rootPath )
      // This will later be obsoleted by the .properties file //
    {
        return
""+rootPath+"/Case_Zo"+str(OutputDataCreater.Zo_OFFSHORE)+"_Cap"+str(OutputDataCreater.CAP)+"/Dia_"+OutputDataCreater.getPM
Group()+"/"+site1+"_"+site2+"_Zo"+str(OutputDataCreater.Zo_OFFSHORE)+"_dp"+str(OutputDataCreater.diaPM)+".dpR";
    }

    private static String str( double in ) // represent a double as an int if their values are equivalent //
    {
        if( in == ((int)in) )
            return ""+((int)in);
        else
            return ""+in;

    }

    /** Static function to return the package name ... */
    public static String getPackageName() {
        return (new Main()).getClass().getPackage().getName();
    }

}
```

TestSummarizer.java –

Creates the seasonal hourly output files and optionally recreates the seasonal output files according to the configuration file and the reomacro file.

```
package Batch2;

/**
* @author  pruibal
 */
import java.io.*;
import java.lang.*;
import java.util.*;


public class TestSummarizer {
    public static final String CONFIG_FILE = Main.CONFIG_FILE;

    public static final boolean REDO_SEASONAL_CALCS = PropertiesViewer.getBoolean( Main.CONFIG_FILE, "RedoSeasonalSummary"
);
    public static final boolean DO_HOURLY_CALCS = PropertiesViewer.getBoolean( Main.CONFIG_FILE, "DoHourlySummary" );

    public static final String MACRO_FILE  = PropertiesViewer.getString(Main.CONFIG_FILE,"OutputSummarizerMacroPath") +
"\\ReOMacro.txt";
    public static final String S_IN_PATH   = PropertiesViewer.getString(Main.CONFIG_FILE,"OutputSummarizerInputPath");


    /** Creates a new instance of TestSummarizer */
    public TestSummarizer() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

    }


    /**
     * Recaluclates all of the seasonal average data
     *  with the option of calculating the annual data based off of the
     *  available seasons
     */
    public static void doSeasonalSummary( ) {
        if( REDO_SEASONAL_CALCS )
        {
            try {
                BufferedReader mreader = new BufferedReader( new FileReader( MACRO_FILE  ));
                int i = Integer.parseInt( mreader.readLine() );


                System.out.print("Re-calculating Seasonal Summary Data...");
                for( int k = 0; k < i; k++ )
                {
```

```
                    String inputFileName = S_IN_PATH + "\\" + mreader.readLine();
                    System.out.print(".");
                    OutputFileSummarizer.doSummary( inputFileName );

                }
                System.out.println("complete!");

            } catch ( Exception e ) {
                System.out.println( "Exception! ");
                e.printStackTrace();
            }
        }

    }
    public static void doHourlySummary( ) {
        if( DO_HOURLY_CALCS )
        {
            try {
                BufferedReader mreader = new BufferedReader( new FileReader( MACRO_FILE ));
                int i = Integer.parseInt( mreader.readLine() );

                System.out.print("Calculating Hourly Summary Data...");
                for( int k = 0; k < i; k++ )
                {
                    String inputFileName = S_IN_PATH + "\\" + mreader.readLine();
                    OutputFileSummarizer.doSummary_Hourly( inputFileName );
                    System.out.print(".");
                }
                System.out.println("complete!");

            } catch ( Exception e ) {
                System.out.println( "Exception! ");
                e.printStackTrace();
            }
        }

    }
}
```

TestSummarySummarizer.java –

Performs all of the summarization algorithms including the surrogate data, hourly summaries, and workbook compilation.

```
package Batch2;
import java.io.*;
import java.util.*;
import java.lang.*;

/**
 *  TestSummarySummarizer - This is the main postprocessing module.
 * This makes all the calls to other modules that check the
 * configuration file and the optionally executes various
 * stages of the postprocessing algorithm (such as hourly summaries,
 * surrogate data, master worksheets, and hourly/seasonal combined
 * workbooks)
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.TestSummarizer
 * @see Batch2.TestDoSurrogate
 */
public class TestSummarySummarizer {
    public static final String CONFIG_FILE = Main.CONFIG_FILE;

    /** Creates a new instance of TestSummarizer */
    public TestSummarySummarizer() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

            // Recreate the summary outputs (at the top of the file) //
            //  if specified by the config file //
        TestSummarizer.doSeasonalSummary();

            // Creates the hourly summary pages, if defined by the config file //
        TestSummarizer.doHourlySummary();

            // Surrogates data into the .source dPr files //
        TestDoSurrogate.main(args);

            // Generate the combined reports //
            // hard-commenting this out overrides the configuration settings //
        generateHourlyCombinedReport( args ); // same as doHourlySummary()
        generateSeasonalCombinedReport( args );
    }

    private static void   generateHourlyCombinedReport(String[] args)
    {
        if( !PropertiesViewer.getBoolean(CONFIG_FILE,"DoHourlySummary") )
            return;
```

```
        try {
            BufferedReader br = new BufferedReader( new FileReader(
              PropertiesViewer.getString(CONFIG_FILE,"SummarizerFileLocation")+"\\SMacro.txt"  ));
            int numRuns, numFiles;
            String[] dprfiles;
            String[] reportsheets;
            String[] titles;
            String pagename, reportTitle, rootPath;

            rootPath = PropertiesViewer.getString(CONFIG_FILE,"SummarizerRootPath");
            numRuns = Integer.parseInt(br.readLine());
            reportsheets = new String[numRuns];
            titles = new String[numRuns];
            for( int i = 0; i < numRuns; i++ )
            {
                numFiles = Integer.parseInt(br.readLine());
                dprfiles = new String[numFiles];
                for( int j = 0; j < numFiles; j++ )
                    dprfiles[j] = convertSlashes(rootPath+"/"+br.readLine()+".hourly" );
                reportsheets[i] = pagename    = convertSlashes(rootPath+"\\"+br.readLine()+".hourly");
                titles[i]       = reportTitle = br.readLine()+"_hourly";

                System.out.print("Generating Hourly Report: \""+reportTitle+"\"...");
                SummarySummarizer.createWorksheet_hourly(dprfiles, pagename, reportTitle );
                System.out.println("Done!");
            }

            String CwN =
convertSlashes(rootPath+"\\"+"hourly_"+PropertiesViewer.getString(CONFIG_FILE,"SummarizerCombinedFile"));
            System.out.print("Generating SummaryWorkbook: \""+CwN+"\"...");


            SummarySummarizer.combineWorksheets_hourly( reportsheets,titles, CwN);
            System.out.println("Done!");

            TestImporter.doHourlySummaryImport();


        } catch( Exception e ) {
            System.out.println("Exception in TestSummarySummarizer.main():\n"+e);
//            e.printStackTrace();
        }
    }

    private static void   generateSeasonalCombinedReport(String[] args)
    {
        if( !PropertiesViewer.getBoolean(CONFIG_FILE,"DoSeasonalSummary") )
            return;
        try {
            BufferedReader br = new BufferedReader( new FileReader(
              PropertiesViewer.getString(CONFIG_FILE,"SummarizerFileLocation")+"\\SMacro.txt"  ));
            int numRuns, numFiles;
            String[] dprfiles;
            String[] reportsheets;
```

```
            String[] titles;
            String pagename, reportTitle, rootPath;

            rootPath = PropertiesViewer.getString(CONFIG_FILE,"SummarizerRootPath");
            numRuns = Integer.parseInt(br.readLine());
            reportsheets = new String[numRuns];
            titles = new String[numRuns];
            for( int i = 0; i < numRuns; i++ )
            {
                numFiles = Integer.parseInt(br.readLine());
                dprfiles = new String[numFiles];
                for( int j = 0; j < numFiles; j++ )
                    dprfiles[j] = convertSlashes(rootPath+"/"+br.readLine());
                reportsheets[i] = pagename    = convertSlashes(rootPath+"\\"+br.readLine());
                titles[i]       = reportTitle = br.readLine();

                System.out.print("Generating Report: \""+reportTitle+"\"...");
                SummarySummarizer.createWorksheet(dprfiles, pagename, reportTitle );
                System.out.println("Done!");
            }

            String CwN = convertSlashes(rootPath+"\\"+PropertiesViewer.getString(CONFIG_FILE,"SummarizerCombinedFile"));
            System.out.print("Generating SummaryWorkbook: \""+CwN+"\"...");
            SummarySummarizer.combineWorksheets( reportsheets,titles, CwN);
            System.out.println("Done!");

            TestImporter.doSeasonalSummaryImport();

        } catch( Exception e ) {
            System.out.println("Exception in TestSummarySummarizer.main():\n"+e);
//            e.printStackTrace();
        }
    }

    private static String convertSlashes( String in )
    {
    int i;
        while( (i=in.indexOf("/")) != -1 )
            in = ""+in.substring(0,i)+"\\"+in.substring(i+1);
        return in;
    }


}
```

TestDoSurrogate.java –

Performs all of the surrogate data calculations, as specified by the Surrogate_Macro file and the configuration file.

```
package Batch2;
import java.io.*;
import java.util.*;
import java.lang.*;



/**
 * TestDoSurrogate - Performs the surrogate data calculation on the
 *  output data files as specified by <code>Surrogate_Macro.txt</code>
 *  in the properties file
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 */
public class TestDoSurrogate {
    public static final String CONFIG_FILE = Main.CONFIG_FILE;




    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        if( !PropertiesViewer.getBoolean(CONFIG_FILE,"DoSurrogate") )
            return;

        try {
            int numRuns;
            String rootPath;

            String srcFile, destFile, impData, useData;

            BufferedReader br = new BufferedReader( new FileReader(
PropertiesViewer.getString(CONFIG_FILE,"SurrogateMacroFileLocation")+"\\Macro_Surrogate.txt"  ));

            rootPath = PropertiesViewer.getString(CONFIG_FILE,"SurrogateRootPath");
            numRuns = Integer.parseInt(br.readLine());

            System.out.println("Inserting Surrogate Data ...");
            for( int i = 0; i < numRuns; i++ )
            {
                srcFile = br.readLine();
                destFile = br.readLine();
                useData = br.readLine();
                impData = br.readLine();

                System.out.print("\t"+useData+":"+srcFile+"...");
                SummarySummarizer.calculateSurrogateData(
convertSlashes(rootPath+"\\"+srcFile),convertSlashes(rootPath+"\\"+destFile), useData, impData );

                if( PropertiesViewer.getBoolean(CONFIG_FILE,"RecalculateAnnualFromSurrogate"))
```

```
                {
                    System.out.print("*...");
                    SummarySummarizer.recalculateAnnualFromSeasons(convertSlashes(rootPath+"\\"+srcFile) );
                }

                System.out.println("Done!");

            }
            System.out.println("Finished Surrogating Data!");

        } catch( Exception e ) {
            System.out.println("Exception in TestDoSurrogate.main():\n"+e);
//            e.printStackTrace();
        }
    }



    private static String convertSlashes( String in )
    {
    int i;
        while( (i=in.indexOf("/")) != -1 )
            in = ""+in.substring(0,i)+"\\"+in.substring(i+1);
        return in;
    }


}
```

TestImporter.java –

Imports a summary page from an external sheet to the combined worksheet from

```
package Batch2;
import java.io.*;
import java.util.*;
import java.lang.*;
import org.apache.poi.hssf.usermodel.*;

/**
 * TestImporter - Imports a specified <code>MasterReportXLSFile</code>
 *  into the Master Report file.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 */
public class TestImporter {
    public static final String CONFIG_FILE = Main.CONFIG_FILE;
    /** Creates a new instance of TestHSSF */
    public TestImporter() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        if( PropertiesViewer.getBoolean(CONFIG_FILE,"DoMasterReportImport") )
        {
            String infile = PropertiesViewer.getString(CONFIG_FILE,"MasterReportXLSFile");
            System.out.print("Importing Master Summary Worksheet from"+infile+"...");
            SheetImporter.importSheet( infile
             , PropertiesViewer.getString(CONFIG_FILE,"MasterSheetName")
             , PropertiesViewer.getString(CONFIG_FILE,"SummarizerRootPath")
                + "\\"+PropertiesViewer.getString(CONFIG_FILE,"SummarizerCombinedFile"));
            System.out.println(" Done!");
        }
    }

    public static void doSeasonalSummaryImport()
    {
        if( PropertiesViewer.getBoolean(CONFIG_FILE,"DoMasterReportImport") )
        {
            String infile = PropertiesViewer.getString(CONFIG_FILE,"MasterReportXLSFile");
            System.out.print("Importing Master Summary Worksheet from"+infile+"...");
            SheetImporter.importSheet( infile
             , PropertiesViewer.getString(CONFIG_FILE,"MasterSheetName")
             , PropertiesViewer.getString(CONFIG_FILE,"SummarizerRootPath")
                + "\\"+PropertiesViewer.getString(CONFIG_FILE,"SummarizerCombinedFile"));
            System.out.println(" Done!");
        }
    }
     public static void doHourlySummaryImport()
    {    }
}
```

Air.java –

Contains a single piece of air information (along with helper functions) for a given record in the output data.

```
package Batch2;

/**
 * Air - Instance of an Air object.  Container class
 *  for reading information about the air, including
 *  temperature, pressure, and humidity for a particular
 *  point in time.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 */
public class Air {

    /** Creates a new instance of Class */
    private double m_tempDegreeC= -6999;
    private double m_pressuremBar= -6999;
    private double m_relativeHumidity= -6999;

    protected boolean dataMissingFlag = true;

    public Air() {
    }

    public Air(double temp, double pressure, double relativeHumidity) {
        m_tempDegreeC = temp;
        m_pressuremBar = pressure;
        m_relativeHumidity = relativeHumidity /100;

        if(temp == -6999 || relativeHumidity == -6999){
            dataMissingFlag = true;
        }
        else{
            dataMissingFlag = false;
        }
    }

    /*********public methods begin********/
    public double getTempInDegreeC(){
        return m_tempDegreeC;
    }
//-------------------------------------------------------------------

    public double getPressuremBar(){
        return m_pressuremBar;
    }
//-------------------------------------------------------------------

    public double getRH(){
        return m_relativeHumidity;
    }
```

```
//-------------------------------------------------------------
    public boolean isDataSet(){
        if(dataMissingFlag == true){
            return false;
        }
        return true;
    }
}
```

AverageAir.java –

Built from a List of Air objects, adds more helper functions for Output

```
package Batch2;
import java.util.*;

/**
 * AverageAir - Container class for calculating the
 *  hourly average air information based off of
 *  an array of Air objects.  This information includes
 *  Virtual Potential temperature and Mixing Ratio
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.Air
 */
public class AverageAir extends Air
{

    private List m_airList = null;

    private double m_avgRH = 0.0;
    private double m_avgTempDegreeC = 0.0;
    private double m_avgPressuremBar = 0.0;

    private int avgCounter = 0;
    private double DEFAULT_PRESSURE = 810.589108713825;
    //private boolean dataMissingFlag = true;

    /** Creates a new instance of AverageAir */
    public AverageAir() {
    }

    public AverageAir(int averageCounter, Air[] a) {
        avgCounter = averageCounter;
        m_airList = Arrays.asList(a);
        calculateAverage();
    }


    /**********public methods begin**************/

    public double getAvgAirPressure(){
        return m_avgPressuremBar;
    }

    public double getAvgRH(){
        return m_avgRH;
    }

    public double getAvgAirTempDegreeC(){
        return m_avgTempDegreeC;
    }

    public double getMixingRatio(){
```

```
        if(isDataSet())
            return 0.623 * m_avgRH * 6.107 * Math.pow(10,(7.5*m_avgTempDegreeC/(237+m_avgTempDegreeC)))/m_avgPressuremBar;
        else
            return 0;
    }

    public double getVirtualPotentialTempDegreeC(){
        if(isDataSet()){
            double w = 0.623 * m_avgRH * 6.107 *
Math.pow(10,(7.5*m_avgTempDegreeC/(237+m_avgTempDegreeC)))/m_avgPressuremBar;
            double Tv = m_avgTempDegreeC *(1 + 0.61 * w);
            double virtualPotentialTemp = Tv * Math.pow((1000 / m_avgPressuremBar), 0.286);
            return virtualPotentialTemp;
        }
        else return 0;
    }

    public double getVirtualPotentialTempDegreeK(){
        return 273.16 + getVirtualPotentialTempDegreeC();
    }

    public void setAirArray(Air[] a){

        m_airList = Arrays.asList(a);
        calculateAverage();
    }

//==========================================================

    /*********private methods begin***************/

    private void calculateAverage(){

        int localCntrForRH = 0;
        int localCntrForTemp = 0;
        int localCntrForPressure = 0;

        ListIterator i = (ListIterator)m_airList.listIterator();
        Air a;
    while( (a = ((Air)i.next()) )!=null ) {
            if( a.getRH()!=-6999){
                m_avgRH += a.getRH();
                localCntrForRH++;
            }
            //else if(a.isDataSet()) m_avgRH = 0;

            if(a.getTempInDegreeC()!=-6999){
                m_avgTempDegreeC += a.getTempInDegreeC();
                localCntrForTemp++;
            }


            if( a.getPressuremBar()!=-6999){
                m_avgPressuremBar += a.getPressuremBar();
```

```
                localCntrForPressure++;
            }
            else {
                m_avgPressuremBar += DEFAULT_PRESSURE;
                localCntrForPressure++;
            }
        }

        if( localCntrForRH == 0 || localCntrForTemp == 0 || localCntrForPressure == 0 )
        {
//            System.out.println("Missing Data: RH-"+localCntrForRH+"  Temp-"+localCntrForTemp+"  Press-
"+localCntrForPressure);
            dataMissingFlag = true;
        }
 /*

        if((0 <= localCntrForTemp && localCntrForTemp < avgCounter ) ||(0 <= localCntrForRH && localCntrForRH < avgCounter
)){
            dataMissingFlag = true;
        }*/
        else{
            m_avgRH /= localCntrForRH;
            m_avgTempDegreeC /= localCntrForTemp;

            if(localCntrForPressure >= avgCounter ){
                m_avgPressuremBar /= localCntrForPressure;
            }
            dataMissingFlag = false;
        }
    }
}
```

Water.java –

Contains a single piece of water information (along with helper functions) for a given record in the output data.

```
package Batch2;

/**
 * Wind - Holds information related to water
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 */
public class Water {

    double m_tempDegreeC = 0.0;

    protected boolean dataMissingFlag = true;

    /** Creates a new instance of Water */
    public Water(){
    }
    public Water(double temp) {
        m_tempDegreeC = temp;
        if(temp == -6999){
            dataMissingFlag = true;
        }
        else {
            dataMissingFlag = false;
        }
    }

//==========================================================

    /***********public methods begin***************/

    public double getTempInDegreeC(){
        return m_tempDegreeC;
    }
//-------------------------------------------------------------
    public boolean isDataSet(){
        if(dataMissingFlag == true){
            return false;
        }
        return true;
    }
}
```

AverageWater.java –

Built from a List of Water objects, adds more helper functions for Output

```
/*
 * AverageWater.java
 *
 * Created on June 17, 2004, 6:31 PM
 */

package Batch2;

import java.util.*;

/**
 * AverageWater - Container class for calculating the
 *  hourly average water information based off of
 *  an array of Water objects.  This information is
 *  only the average water temperature.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.Water
 */
public class AverageWater extends Water {

    private double m_avgTempDegreeC = 0.0;
    private List m_waterList = null;

    private int avgCounter = 0;
    private double DEFAULT_TEMPERATURE = 13.667041632946704;

    /** Creates a new instance of AverageWater */
    public AverageWater(){
    }

    public AverageWater(int averageCounter, Water[] waterArray){

        avgCounter = averageCounter;
        m_waterList = Arrays.asList(waterArray);
        calculateAverage();
    }
//==========================================================

    /**************public methods begin*****************/
    private static final double waterTempMod = PropertiesViewer.getDouble(Main.CONFIG_FILE,"WaterTempAdjustment");

    public double getAvgTempDegreeC(){
        return m_avgTempDegreeC;
    }

    /***************private methods begin****************/
    private void calculateAverage(){

        int localCntr = 0;
        Water w = null;
```

```
        //double tempRH = 0.0;
        //double tempTempDegreeC = 0.0;
        //double tempPressuremBar = 0.0;

        ListIterator i = (ListIterator)m_waterList.listIterator();
    while((w = (Water)i.next())!=null ){
            if(w.isDataSet() && w.getTempInDegreeC() != -6999){
                m_avgTempDegreeC += w.getTempInDegreeC();
                localCntr++;
            }
            else{//if data is not set
                m_avgTempDegreeC += DEFAULT_TEMPERATURE;
                localCntr++;
            }
        }
        w = null;
        if(0<= localCntr && localCntr < avgCounter ){
            dataMissingFlag = true;
        }
        else{
            m_avgTempDegreeC /= localCntr;
            m_avgTempDegreeC += waterTempMod;     // THIS INSERTS THE H2O BIAS //
            dataMissingFlag = false;
        }
    }
}
```

Wind.java –

Contains a single piece of Wind information (along with helper functions) for a given record in the output data.

```
package Batch2;
import java.io.*;
import java.lang.*;
import java.lang.Math;

/**
 * Wind - Holds information related to wind
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 */
public class Wind {

    private double m_speedInMPS = 0;
    private double m_directionInDegrees = 0;

    protected boolean dataMissingFlag = true;


    /** Creates a new instance of Wind */
    public Wind() {
    }

    public Wind(double windSpeedInMetersPerSecond, double windDirection){
            m_speedInMPS = windSpeedInMetersPerSecond;
            m_directionInDegrees = windDirection;

        if(windSpeedInMetersPerSecond == -6999 && windDirection == -6999){
            dataMissingFlag = true;
        }
        else{
            dataMissingFlag = false;
        }
    }


    //public methods
    public void setWindDirectionInDegrees(double wind_Direction) {
        m_directionInDegrees = wind_Direction;
        dataMissingFlag = false;
    }

    public void setWindSpeedInMetersPerSecond(double wind_speed_in_meters_per_second) {
        m_speedInMPS = wind_speed_in_meters_per_second;
        dataMissingFlag = false;
    }

    public void setWindSpeedInKnots(double wind_speed_in_knots) {
        m_speedInMPS = (wind_speed_in_knots/1.943846);
    }
```

```
    public double getWindDirectionInDegrees() {
        return m_directionInDegrees;
    }

    public double getWindSpeedInMetersPerSecond() {
        return m_speedInMPS;
    }

    public double getWindSpeedInKnots() {
        return (m_speedInMPS * 1.943846);
    }

    public boolean isDataSet(){
        if(dataMissingFlag == true){
            return false;
        }
        return true;
    }
}
```

AverageWind.java –

Built from a List of Wind objects, adds more helper functions for Output

```
/*
 * AverageWind.java
 *
 * Created on June 8, 2004, 6:37 PM
 */

package Batch2;

import java.util.*;
import java.util.List;

/**
 * AverageWind - Container class for calculating the
 *  hourly average wind information based off of
 *  an array of Wind objects.  This information includes
 *  the WDQI, Vd from Bulk Coefficent, Z0, etc..
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.Wind
 */
public class AverageWind extends Wind{

    private double m_averageVectorWindSpeedInKnots = 0;
    private double m_vectorDirectionInTensOfDegrees = 0;
    private double m_depositionVelocity = 0;

    private List m_windList = null;

    /************this variable is the indicator of whether the wind is
     offshore: from land to water/ (+1)
     onshore: from water to land/ (-1)
     sideshore:(0)    +/- 20 degree of parallel to showerline etc.
     (values are: +1, -1, 0 respectively)
    ***********/
    private int m_windDirQualityIndex = 2;/*invalid initial value*/
    private double beta = 30; // this is the angle used for the measurement of sideshore wind
    //private boolean dataMissingFlag = true;

    private Station stTemp = null;
    private double summationU = 0.0, summationV = 0.0;
    private int avgCounter = 0;

    private static final double speedMultiplier = PropertiesViewer.getDouble(Main.CONFIG_FILE,"WindSpeedMultiplier");

    /** Creates a new instance of AverageWind */
    public AverageWind() {
    }

    public AverageWind(int averageCounter, Wind[] windArray){

        avgCounter = averageCounter;
```

```
        m_windList = Arrays.asList(windArray);
        calculateAverage();
    }

    public AverageWind(int averageCounter, Wind[] windArray, Station st){

        avgCounter = averageCounter;
        m_windList = Arrays.asList(windArray);
        stTemp = st;
        calculateAverage();
        calculateWDQI();
    }

    /*******public methods begin*************/

    public void setWindArray(Wind[] windArray){

        m_windList = Arrays.asList(windArray);
        calculateAverage();
    }
    //-----------------------------------------------------------------------

    public double getWindDirectionInDegrees() {
        return m_vectorDirectionInTensOfDegrees *10;
    }
    //-----------------------------------------------------------------------
    public double getWindDirectionInTensOfDegrees() {
        return m_vectorDirectionInTensOfDegrees;
    }
//-----------------------------------------------------------------------
    public double getWindSpeedInMetersPerSecond() {
        return m_averageVectorWindSpeedInKnots / 1.943846;
    }
//-----------------------------------------------------------------------
    public double getWindSpeedInKnots() {
        return m_averageVectorWindSpeedInKnots;
    }
//-----------------------------------------------------------------------------
    public int getWDQI(){
        return m_windDirQualityIndex;
    }
//-----------------------------------------------------------------------

    public double getDepositionVelocityFromBulkTransferCOeff(){
        double windSpeedMPS = getWindSpeedInMetersPerSecond();

        if(windSpeedMPS == 0){
            return m_depositionVelocity;
        }


        double bulkTransferCOeff = 0; /****Cu n*****/
        double frictionalVelocity = 0; /*****uStar*******/
```

```
        bulkTransferCOeff = (0.75 + 0.067*windSpeedMPS)/1000;

        frictionalVelocity = windSpeedMPS * Math.sqrt(bulkTransferCOeff);
        m_depositionVelocity =(frictionalVelocity/ windSpeedMPS) * frictionalVelocity *100;

        return m_depositionVelocity;
    }

    public double getDepositionVelocityFromBulkTransferCOeff( double myU10 ){
        double windSpeedMPS = myU10;

        if(windSpeedMPS == 0){
            return m_depositionVelocity;
        }


        double bulkTransferCOeff = 0; /****Cu n*****/
        double frictionalVelocity = 0; /*****uStar*******/

        bulkTransferCOeff = (0.75 + 0.067*windSpeedMPS)/1000;

        frictionalVelocity = windSpeedMPS * Math.sqrt(bulkTransferCOeff);
        m_depositionVelocity =(frictionalVelocity/ windSpeedMPS) * frictionalVelocity *100;

        return m_depositionVelocity;
    }

//-------------------------------------------------------------------------



//=========================================================================


    /*******private methods begin**********/


    /*************this method is called by  getDepositionVelocityFromStationHeight() method********************/
    private double calculateZo(){
        double windSpeedMPS = getWindSpeedInMetersPerSecond();

        int wdqi = getWDQI();
        if(wdqi  == 1)
            return OutputDataCreater.Zo_OFFSHORE;
            // return 0.1;
        if(wdqi == -1)
            return 0.000002 * Math.pow(windSpeedMPS, 2.5);
        if(wdqi == 0)
            return (OutputDataCreater.Zo_OFFSHORE + 0.000002 * Math.pow(windSpeedMPS, 2.5))/2;

            //return (0.1 + 0.000002 * Math.pow(windSpeedMPS, 2.5))/2;
        return 0;
```

```
    }
//-------------------------------------------------------------------------
    private void calculateWDQI(){
        m_windDirQualityIndex = stTemp.getWDQIchecked((m_vectorDirectionInTensOfDegrees *10) % 360);
    }


//-----------------------------------------------------------------
    private void calculateAverage(){

        int actualCntr = calculateSummations();

        if(actualCntr >= avgCounter){
            m_vectorDirectionInTensOfDegrees = calculateVectorWindDirectionInTensOfDegrees();
            m_averageVectorWindSpeedInKnots = calculateVectorWindSpeedInKnots(actualCntr)*speedMultiplier;
            dataMissingFlag = false;
        }
        else{
            m_vectorDirectionInTensOfDegrees = 0;
            m_averageVectorWindSpeedInKnots = 0;
            dataMissingFlag = true;
        }
    }
    //-------------------------------------------------------------------------

    private double calculateVectorWindDirectionInTensOfDegrees(){
        double arcTangentValue = 0.0;
        if(summationV==0){
            if(summationU<0){
                arcTangentValue = -Math.PI/2;
            }
            else if(summationU>0){
                arcTangentValue = Math.PI/2;
            }
        }
        else{
            arcTangentValue = Math.atan(summationU/summationV);
        }
        return (Math.toDegrees(arcTangentValue) + getTheta())/10;
    }
    //-----------------------------------------------------
    private double calculateVectorWindSpeedInKnots(int avgCntr){

        /*
        int windArraySize = 0;
        if(m_windList.indexOf(null)!= -1 ){
            windArraySize= m_windList.indexOf(null);
        }
        else windArraySize = m_windList.size();
        */
        return ((Math.sqrt(summationU*summationU + summationV*summationV))/avgCntr) * 1.943846;
    }
```

```
    //-------------------------------------------------------
    private double getTheta(){
        if(summationV>=0) return 180;
        if(summationU<0 && summationV<0) return 0;
        if(summationU>=0 && summationV<0) return 360;
        return 0;
    }
    //-------------------------------------------------------------------
   private int calculateSummations(){

        int localCntr = 0;
    Wind w= null;

        double accuracyValue = 0.0000000001;
        ListIterator i = (ListIterator)m_windList.listIterator();

        while((w = (Wind)i.next())!=null ){
            if(w.isDataSet()){
                double tempSine = Math.sin(Math.toRadians(w.getWindDirectionInDegrees()));
                double tempCosine = Math.cos(Math.toRadians(w.getWindDirectionInDegrees()));

                if(Math.abs(tempSine)<=accuracyValue){
                    tempSine = 0.0;
                }

                if(Math.abs(tempCosine)<=accuracyValue){
                    tempCosine = 0.0;
                }

                summationU += (-w.getWindSpeedInMetersPerSecond()*tempSine);
                summationV += (-w.getWindSpeedInMetersPerSecond()*tempCosine);
                localCntr++;
            }
        }

        w = null;
        return localCntr;
    }
}
```

Station.java –

Contains Station specific information, such as sensor constants and functionality for a particular site/station combination.

```
package Batch2;


 import java.util.Arrays;
 import java.util.ArrayList;
 import java.util.Collections;
 import java.util.GregorianCalendar;
 import java.util.Iterator;
 import java.util.ListIterator;
 import java.util.List;


/**
 * Station - Holds information and constants concerning the met stations.
 *  This is very important in determining the WDQI wind values, and in
 *  generating output files.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 */
 public final class Station implements Comparable, java.io.Serializable
 {
      /** Default Cave Rock Met Pair/Station info */
   public static final Station CAVE_ROCK =   new Station("CAVE ROCK", "TB"  , 13103, 7,   10,  190);
      /** Default Rubicon Met Pair/Station info */
   public static final Station RUBICON =     new Station("RUBICON", "TB"    , 12103, 5.7, 160, 340);
      /** Default Sunnyside Met Pair/Station info */
   public static final Station SUNNYSIDE =   new Station("SUNNYSIDE", "LF"  , 12303, 5.7, 205, 25);
      /** Default Timber Cove Met Pair/Station info */
   public static final Station TIMBER_COVE = new Station("TIMBER COVE", "SW", 12403, 7,   60,  240);
      /** Default TDR1 Buoy Met Pair/Station info */
   public static final Station TDR1 =        new Station("TDR1", "SW"       , 11103, 3,   60,  240);
      /** Default TDR2 Buoy Met Pair/Station info */
   public static final Station TDR2 =        new Station("TDR2", "SW",        11203, 3,   60,  240);
      /** Default Tahoe Vista Met Pair/Station info */
   public static final Station TAHOE_VISTA = new Station("TAHOE VISTA", "LF", 12503, 6.7, 279, 99);
      /** Default US Coast Guard Met Pair/Station info */
   public static final Station USCG =        new Station("USCG", "LF",        12603, 6.5, 255, 75);

      /** Array of default stations */
   private static final Station[] STATION_ARRAY =
       { CAVE_ROCK, RUBICON, SUNNYSIDE, TIMBER_COVE, TDR1, TDR2, TAHOE_VISTA, USCG };
      /** Array of default stations */
   public static final List STATIONS = Collections.unmodifiableList(Arrays.asList(STATION_ARRAY));

   // count will be seen as zero despite its value unless it's before the
   // public constant declarations
   private static int count = 0;
   private static double beta = 30;
```

```
    private String m_name = ""; //Station no. etc.
    private String m_corresStation = "";  // this the station where the PM and gas related readings are taken
    private int m_number = -1;  // 5 digit etc.
    private double m_sensorHeight = -1;    //

    private double m_leftAngle = -1;    // 0 to 360 etc.
    private double m_rightAngle = -1;    // 0 to 360 etc.

    private double lowerLimitAngle = -1;    // 0 to 360 etc.
    private double upperLimitAngle = -1;    // 0 to 360 etc.

    private int particularIndex = -1;


    /** Public constructor, given a Met site and an AQ site.
     * The constructor attempts to match an internal site with
     * the one provided.
     *@param stName The station name for MET data
     *@param corres The AQ site (for output purposes)
     */
    public Station(String stName, String corres)
    {
        ListIterator i = (ListIterator)STATIONS.listIterator();
        while(i.hasNext())
        {
            String strTemp = null;
            Station st = null;
            st = (Station)i.next();
            strTemp = st.getName();
            if(stName.equals(strTemp))
            {
                this.m_name = st.getName();
                this.m_corresStation = corres;
                this.m_number = st.getNumber();
                this.m_sensorHeight = st.getSensorHeight();
                this.m_leftAngle = st.getLeftAngle();
                this.m_rightAngle = st.getRightAngle();
                particularIndex = i.previousIndex();
                return;
            }
        }
        System.out.println("Station not found!");
        System.exit(1);
    }


        /************private constructor**************/
        /* If WD is equal to leftAngle then with wind at your back the wind is parallel to shore
         and the shore is on your left water is on the right.
         Similarly for rightAngle, if WD is equal to right angle then with the wind at your back then
         land is on your right and water is on your left.
         from leftAngle clockwise to rightAngle all included WD values represent offshore flow.************/
    private Station(String name, String corresStation, int number, double sensorHeight, double leftAngle, double
rightAngle)
```

```
    {
        m_name = name;
        m_corresStation = corresStation;
        m_number = number;
        m_sensorHeight = sensorHeight;
        m_leftAngle = leftAngle % 360;
        m_rightAngle = rightAngle % 360;
        count++;
    }
//==================================================

    /**************public methods*****************/

    public String getName() {
            return m_name;
    }


    public String getCorresStation(){
        return m_corresStation;
    }

    public int getNumber() {
        return m_number;
    }

    public double getSensorHeight(){
        return m_sensorHeight;
    }

    public double getLeftAngle() {
        return m_leftAngle;
    }

    public double getRightAngle() {
        return m_rightAngle;
    }

        /*******this method returns the lower limit angle for calculating offshore********/
    public double getLowerLimitForOffshore(){
        return calculateLimitAngle(m_leftAngle + beta/2);
    }

        public double getUpperLimitForOffshore(){
            double upperLimit = calculateLimitAngle(m_rightAngle - beta/2);
            if(upperLimit < calculateLimitAngle(m_leftAngle + beta/2)){
                upperLimit += 360;
            }
            return upperLimit;
        }

        public double getLowerLimitForRightSideshore(){
            return calculateLimitAngle(m_rightAngle - beta/2);
```

```
        }

        public double getUpperLimitForRightSideshore(){
            double upperLimit = calculateLimitAngle(m_rightAngle + beta/2);
            if(upperLimit < calculateLimitAngle(m_rightAngle - beta/2)){
                upperLimit += 360;
            }
            return upperLimit;
        }

        public double getLowerLimitForOnshore(){
            return calculateLimitAngle(m_rightAngle + beta/2);
        }

        public double getUpperLimitForOnshore(){
            double upperLimit = calculateLimitAngle(m_leftAngle - beta/2);
            if(upperLimit < calculateLimitAngle(m_rightAngle + beta/2)){
                upperLimit += 360;
            }
            return upperLimit;
        }

        public double getLowerLimitForLeftSideshore(){
            return calculateLimitAngle(m_leftAngle - beta/2);
        }

        public double getUpperLimitForLeftSideshore(){
            double upperLimit = calculateLimitAngle(m_leftAngle + beta/2);
            if(upperLimit < calculateLimitAngle(m_leftAngle - beta/2)){
                upperLimit += 360;
            }
            return upperLimit;
        }

    public String toString() {
            return m_name;
    }

    public int getWDQIchecked(double angle)
    {
        if(isInRange(getLowerLimitForOffshore(), angle, getUpperLimitForOffshore() )){
            return +1;//offshore
        }
        else if(isInRange(getLowerLimitForRightSideshore(), angle, getUpperLimitForRightSideshore())){
            return 0;//sideshore
        }
        else if(isInRange(getLowerLimitForOnshore(), angle, getUpperLimitForOnshore())){
            return -1;//onshore
        }
        else if(isInRange(getLowerLimitForLeftSideshore(), angle, getUpperLimitForLeftSideshore())){
            return 0;//sideshore
        }
        return 2;
```

```
    }

    public int compareTo(Object o) {
        return getName().compareTo(((Station) o).getName());
    }


//=================================================================
        /********private methods begin*************/

    private double calculateLimitAngle(double angle)
    {
        angle = angle % 360;
        if(0 <= angle)
            return angle;

        return 360 + angle;
     }
//-----------------------------------------------------------------
    /** Used to decide whether the angle is in the
     range of lowerLimit and lowerLimit+upperOffset********/
    private boolean isInRange(double lowerLimit,double angle, double upperLimit){
        angle = angle % 360;
        while(angle < upperLimit){
            if(lowerLimit <= angle && angle < upperLimit)
                return true;
            angle += 360;
        }
        return false;
    }
}
```

MyGregorianCalendar.java –

Extension of java's GregorianCalendar class, used to convert from Julian to Gregorian time, adjusting the time zone, also able to return the season name.

```
package Batch2;
import java.util.*;

/**
 * MyGregorianCalendar - An extension of the java standard
 *  GregorianCalendar class with added season-handling
 *  parsing any number of input files, doing summaries of
 *  the output data, generating excel spreadsheets,
 *  surrogating output data, and importing a master
 *  report.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.TestSummarySummarizer
 * @see Batch2.OutputFileSummarizer
 */
public class MyGregorianCalendar extends GregorianCalendar {

    /** Creates a null instance of MyGregorianCalendar */
    public MyGregorianCalendar() {
    }

    /** Creates an instance from input data, validates time, adjusts to PST. */
    public MyGregorianCalendar(String year, String yearInHeader, String day, String time){
        int hour = -1, minute = -1;


                    /***check if the year previously given on second line----
             ----- and that read on this line matches****/
        if(!year.equalsIgnoreCase(yearInHeader)){
            System.out.println("Years in header and in data don't match....");
            System.out.println("exiting abnormally.....");
            System.exit(1);/**method terminated abnormally**/
        }
                    /***check if the days read on each line exceed 366 ****/
        if(Integer.parseInt(day)>366){
            System.out.println("Days exceed normal days in Year.....");
            System.out.println("exiting abnormally.....");
            System.exit(1);/**method terminated abnormally**/
        }

        /**check if time is a valid time ***/
        if(time.length()<=2){
            if(0<=Integer.parseInt(time) && Integer.parseInt(time)<=59 ){
                minute = Integer.parseInt(time);
                hour = 0;
            }
            else{
                System.out.println("Time is not a valid time.....");
                System.out.println("exiting abnormally.....");
```

```
                System.exit(1);
            }
        }
        else if(time.length()<=4){
            hour = Integer.parseInt(time)/100;
            minute = Integer.parseInt(time)%100;
            if(!(0<=minute && minute<=59) ){
                System.out.println("Time is not a valid time.....");
                System.out.println("exiting abnormally.....");
                System.exit(1);
            }
            if(!(1<=hour && hour<=23) ){
                System.out.println("Time is not a valid time.....");
                System.out.println("exiting abnormally.....");
                System.exit(1);
            }
        }

        /*********create a timezone with GMT and create calendar with that
         because the times we have are all GMT and then convert it to PST when
         we have initialized with the year, month, day, time etc.****************/

//        SimpleTimeZone tz = new SimpleTimeZone(0,"GMT");
        //TimeZone tz = SimpleTimeZone.getTimeZone("GMT");
//        super.setTimeZone(tz);
        //GregorianCalendar cal= new GregorianCalendar();
        super.set(GregorianCalendar.YEAR, Integer.parseInt(year));
        super.set(GregorianCalendar.DAY_OF_YEAR, Integer.parseInt(day));
        super.set(GregorianCalendar.HOUR_OF_DAY, hour);
        super.set(GregorianCalendar.MINUTE, minute);


        /************just call a get method to make sure the previously set fields
         are actually set*****************/
//        super.get(GregorianCalendar.MINUTE);


        /***********change the timezone to PST
         so that it converts the time to 8 hrs earlier************/
        // DOESN'T WORK
//        super.setTimeZone(new SimpleTimeZone(-8*60*60*1000, "America/Los_Angeles"));

            // THIS DOES WORK //
        super.add( HOUR_OF_DAY, -8 );

    }


    /** Returns the season based on the month */
    public String getSeason(){
        int mnth = super.get(GregorianCalendar.MONTH);
        switch( mnth )
        {
```

```
        case 11: case 0: case 1:
            return "Winter";
        case 2: case 3: case 4:
            return "Spring";
        case 5: case 6: case 7:
            return "Summer";
        case 8: case 9: case 10:
            return "Fall";
        default:
            return "";
        }
    }
}
```

ReaderDriver.java –

Parses a Compiled hourly data file, determining data columns from a user input stream (either stdin or macro.txt). Capable of creating OutputDataCreater objects.

```
package Batch2;
import java.io.*;
import java.lang.*;
import java.util.GregorianCalendar;
import java.util.SimpleTimeZone;

/**
 * ReaderDriver - Class for processing
 *  the input Met files... This action could be broken
 *  into two separate operations, in the future.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.SummarySummarizer
 * @see Batch2.PropertiesViewer
 */
 public class ReaderDriver {

    /****private variables for file handling and logging and counting*****/

    private BufferedReader m_input= null;
    private String prevLineString = null;
    private int m_averageCounter  = -1;

    /**private variables of the class*/
   // private int AVERAGE_TIME_FOR_WIND_READING = 10;/*IT'S 10 min currently**/
    //private int LOOPING_COUNTER_FOR_HOUR = 60/AVERAGE_TIME_FOR_WIND_READING;
   // private int STARTING_MINUTE_OF_HOUR = 0;

    /**used to be inserted to CD144 object*****/
    private String  stationNumber       = "";
    private Station stationInstance     = null;

    /********for comparison and validation purpose********/
    private String yearInHeader = "";

    /***variables needed to be used as array index***/
    private int yearIndex = -1;
    private int dayIndex = -1;
    private int timeIndex = -1;

    private int vectorWindSpeedIndex = -1;
    private int windDirectionIndex = -1;

    private int airTempIndex = -1;
    private int airPressureIndex = -1;
    private int airRHIndex = -1;

    private int waterTempIndex = -1;
```

```
/** Creates a new instance of ReaderDriver */
public ReaderDriver() {
}

private BufferedReader userInput;
public ReaderDriver(String inputFile, int avgCnt, BufferedReader br ) {
     try {
        //m_errorStream = new PrintWriter(new FileOutputStream(LOG_FILE));
        m_averageCounter = avgCnt;
        m_input = new BufferedReader(new FileReader(inputFile));
        userInput = br;
    }
    catch(Exception e) {
         reportError(e);
         //closeAllStreams();
     }
}

/********public methods begin**********/

public void parseInputFileForVariableCollection( String AQ_Site ){
    /**temporary variables***/
    String tempString = "";
    String tempStringLower = "";
    String[] tempArray = null;
    boolean MACRO_MODE = Main.MACRO_MODE;


    /**file handler variables***/
    BufferedReader br = null;



            /*****************************************
             *read the station name on the first line
             *****************************************            */
    try{
    tempString = m_input.readLine();
    tempStringLower = tempString.toLowerCase();
    if(!(boolean)(tempStringLower.indexOf("station")<0)){
        tempArray = tempString.split(":");
        //System.out.print("Please enter the Station # for "+tempArray[1].trim()+" (has to be 5 digits)"+": ");

        /****create a station instance with the station name found in file******/
        stationInstance = new Station(tempArray[1].trim() , AQ_Site  );
    }
    else{
        reportError("station name. not found on the first line");
        System.out.println("exiting abnormally....");
        System.exit(1);/**method terminated abnormally**/
    }
```

```
        /**********************************************
         *read the year on the second line
         **********************************************
         */
tempString = m_input.readLine();
tempStringLower = tempString.toLowerCase();
if(!(boolean)(tempStringLower.indexOf("year")<0)){
    tempArray = tempString.split(":");
            // validation checks are needed
    yearInHeader = tempArray[1].trim();
}
else{
    reportError("year not found on the second line");
    System.out.println("exiting abnormally....");
     System.exit(1);/**method terminated abnormally**/
}

        /*************************************************************
         *pass thro' several lines till you reach the first line of
         *the table data
         *************************************************************
         */
do{
    tempString = m_input.readLine();
    tempArray =  tempString.split("\\s");
}while(!(isInArray(tempArray, "year") && isInArray(tempArray, "time") &&isInArray(tempArray, "day")  ));


System.out.println("I have found the following variables while scanning the file...");

try{
    for(int i = 0; i<tempArray.length; ){
        System.out.println(i+"\t"+tempArray[i++]+"\t\t\t"+i+"\t"+tempArray[i++]);
    }
}catch(ArrayIndexOutOfBoundsException e){}

    System.out.println("Please enter the # of variable above, that you want to assign");
    System.out.print("as Year: ");

    br = userInput;

    yearIndex = Integer.parseInt(br.readLine()); //can throw exception
    if( MACRO_MODE ) System.out.println(""+yearIndex);

    System.out.println("Please enter the # of variable above, that you want to assign");
    System.out.print("as Day: ");
    dayIndex = Integer.parseInt(br.readLine()); //can throw exception
    if( MACRO_MODE ) System.out.println(""+dayIndex);

    System.out.println("Please enter the # of variable above, that you want to assign");
    System.out.print("as Time: ");
    timeIndex = Integer.parseInt(br.readLine()); //can throw exception
```

```
            if( MACRO_MODE ) System.out.println(""+timeIndex);

            System.out.println("Please enter the # of variable above, that you want to assign");
            System.out.print("as \"Analog Wind Speed Vector Value (meter/secs) \": ");
            vectorWindSpeedIndex = Integer.parseInt(br.readLine()); //can throw exception
            if( MACRO_MODE ) System.out.println(""+vectorWindSpeedIndex );

            System.out.println("Please enter the # of variable above, that you want to assign");
            System.out.print("as \"Analog Wind Speed Direction Value (degrees) \": ");
            windDirectionIndex = Integer.parseInt(br.readLine()); //can throw exception
            if( MACRO_MODE ) System.out.println(""+windDirectionIndex );

            System.out.println("Please enter the # of variable above, that you want to assign");
            System.out.print("as \"Air Temperature in Degree C\": ");
            airTempIndex = Integer.parseInt(br.readLine()); //can throw exception
            if( MACRO_MODE ) System.out.println(""+airTempIndex );

            System.out.println("Please enter the # of variable above, that you want to assign");
            System.out.print("as \"Air Pressure in milliBars\": ");
            airPressureIndex = Integer.parseInt(br.readLine()); //can throw exception
            if( MACRO_MODE ) System.out.println(""+airPressureIndex );

            System.out.println("Please enter the # of variable above, that you want to assign");
            System.out.print("as \"Relative Humidity of air \": ");
            airRHIndex = Integer.parseInt(br.readLine()); //can throw exception
            if( MACRO_MODE ) System.out.println(""+airRHIndex );

            System.out.println("Please enter the # of variable above, that you want to assign");
            System.out.print("as \"Temperature of Water \": ");
            waterTempIndex = Integer.parseInt(br.readLine()); //can throw exception
            if( MACRO_MODE ) System.out.println(""+waterTempIndex);

        System.out.println("Busy...Please wait!!!");
        }
        catch(Exception e){
            reportError(e);
            System.out.println(""+e);
            closeAllStreams();
            System.exit(1);
        }


    }/**end of method parseInputFile()**/
    //-------------------------------------------------------------------
```

```
    public static final String WATER_TEMP_FILENAME  = PropertiesViewer.getString(Main.CONFIG_FILE,"WaterTempFileName");
    public static final String AIR_TEMP_FILENAME    = PropertiesViewer.getString(Main.CONFIG_FILE,"AirTempFileName");

    private static AvgWaterTempReader joesWaterData = new AvgWaterTempReader( WATER_TEMP_FILENAME );
    private static AvgAirTempReader   joesAirData   = new AvgAirTempReader( AIR_TEMP_FILENAME );

    private static final int MAX_DATA_POINTS = 30;
    private double prev_water = -6999.0;
    private double prev_air   = -6999.0;

    public OutputDataCreater createODCfromInput()
    {
        try {
            int     counter = 0;
            Wind[]  windArray  = new Wind[MAX_DATA_POINTS];
            Air[]   airArray   = new Air[MAX_DATA_POINTS];
            Water[] waterArray = new Water[MAX_DATA_POINTS];

            MyGregorianCalendar current = null;
            MyGregorianCalendar old = null;
            String s;

            m_input.mark( 3*250 ); // set a io mark //
            while( true )
            {
                String currentRecord   = m_input.readLine();         // get the current record

                String currentFields[] = currentRecord.split("\\t"); // Create array of fields from current record

                current = new MyGregorianCalendar(currentFields[yearIndex], yearInHeader, currentFields[dayIndex],
currentFields[timeIndex]);

                    // CONVERT TO PST //
//                current.set( GregorianCalendar.HOUR_OF_DAY, (current.get( GregorianCalendar.HOUR_OF_DAY ) + 16)%24 );

                // remove this line //
//                System.out.println("Time from File: "+currentFields[timeIndex]+"  Hour from calendar: "+
current.get(MyGregorianCalendar.HOUR_OF_DAY )
//                 + "  Should be: "+ (( current.get(MyGregorianCalendar.HOUR_OF_DAY )+16)%24))  ;



                int year = current.get( GregorianCalendar.YEAR ),
                    month= current.get( GregorianCalendar.MONTH )+1,
                    day  = current.get( GregorianCalendar.DAY_OF_MONTH ),
                    hour = current.get( GregorianCalendar.HOUR_OF_DAY );


                if( old != null ) // Check to make sure data still applies to this set  //
```

```
                    if( hour != old.get(GregorianCalendar.HOUR_OF_DAY) )
                        break;

                old=current;

                    // After Current Record has been Validated ...
                m_input.mark( 3*200 );  // set a new mark //


                    /***  AIR   ***/
                if( (s = validateAndCollectData(airTempIndex, currentFields)).compareToIgnoreCase("-6999") == 0 )
                    s = "" + joesAirData.getData(year, month,day,hour);

                airArray[counter] = new Air(
                    Double.parseDouble(s),
                    Double.parseDouble(validateAndCollectData(airPressureIndex, currentFields)),
                    Double.parseDouble(validateAndCollectData(airRHIndex, currentFields))
                );

                    /***  WATER  ***/
                if( (s = validateAndCollectData(waterTempIndex, currentFields)).compareToIgnoreCase("-6999") == 0 )
                    s = "" + joesWaterData.getData(year, month,day,hour);

                    // Add a new water data point to the current set //
                waterArray[counter] = new Water( Double.parseDouble(s) );

                    /***  WIND  ***/
                windArray[counter] = new Wind(
                    Double.parseDouble(currentFields[vectorWindSpeedIndex]),
                    Double.parseDouble(currentFields[windDirectionIndex])
                );
                counter++;
            }
            m_input.reset();  //reset the BufferedReader to the previous entry

            // Create the OutputDataCreater object //
            if( counter < m_averageCounter ) {
//                System.out.println(""+counter+" vs. "+m_averageCounter);
                return new OutputDataCreater( stationInstance, old );
            } else {
                // Create the AverageAir, AverageWind, AverageWater objects //
                AverageAir   a1 = new AverageAir(counter,airArray);
                AverageWind  a2 = new AverageWind(counter,windArray, stationInstance);
                AverageWater a3 = new AverageWater(counter,waterArray );
                return new OutputDataCreater(stationInstance, old, a2, a1, a3);
            }
        } catch( Exception e ) {
//            System.out.println("Exception createODCfromInput()");
//            e.printStackTrace();
            return null;
//            System.exit(0);
        }
//        return null;
```

```
    }

    public OutputDataCreater collectDataForCD144() /** DEPRECATED **/
// Creates on OutputDataCreater object based on the current line of information
// in the main input file.
    {

        /**variables used to hold the line from file and separated fields in dataArray ***/
        String dataLineString ="";
        String[] dataArray=null;

        /***variables needed by Output format and by calendar and wind classes---
         ---only taken that are necessary here ****/
        String year = "";
        String day = "";
        String time = "";

        String windDirection ="";
        String windSpeed ="";

        String airTemp = "";
        String airPressure ="";
        String airRH = "";

        String waterTemp = "";

        /**calendar and wind array needed to be passed in-----
         -----the CD144 class for writing to the file**/
        //GregorianCalendar[] calendarArray = new GregorianCalendar[LOOPING_COUNTER_FOR_HOUR];

            // the maximum # of points for an average is 15 //
        Wind[]  windArray  = new Wind[15];
        Air[]   airArray   = new Air[15];
        Water[] waterArray = new Water[15];

        /****************************************/
        MyGregorianCalendar prevCalendar = null;
        MyGregorianCalendar nextCalendar = null;

        OutputDataCreater cd = null;

        int localCounter = -1;
        try{

              /*
               * PERFORM ACTIONS ON THE FIRST PEICE OF DATA FOR THE HOUR
               */
            if(prevLineString != null)
                dataLineString = prevLineString;     // get last data from previous iteration
            else
                dataLineString = m_input.readLine(); // get the first line of data

            dataArray = dataLineString.split("\\t"); // Create a 1-d array (vector) from current entry
```

```
            year = dataArray[yearIndex];  // GET YEAR from data file / user input
            day  = dataArray[dayIndex];   // GET DAY from data file / user input
            time = dataArray[timeIndex];  // GET TIME from data file / user input

                /*****create a calendar for the time identification purpose*********/

            prevCalendar = new MyGregorianCalendar(year, yearInHeader, day, time);


            windDirection = dataArray[windDirectionIndex];
            windSpeed     = dataArray[vectorWindSpeedIndex];
            windArray[++localCounter] = new Wind(Double.parseDouble(windSpeed), Double.parseDouble(windDirection));

            airTemp     = validateAndCollectData(airTempIndex, dataArray);//  dataArray[airTempIndex];
             /* IF AIR TEMP IS BAD, GET FROM TABLE */
            airPressure = validateAndCollectData(airPressureIndex, dataArray);// dataArray[airPressureIndex];
            airRH       = validateAndCollectData(airRHIndex, dataArray);// dataArray[airRHIndex];
            airArray[localCounter] = new Air(Double.parseDouble(airTemp),
Double.parseDouble(airPressure),Double.parseDouble(airRH));

            waterTemp = validateAndCollectData(waterTempIndex, dataArray);
             /* IF WATER TEMP IS BAD, GET FROM TABLE */
            waterArray[localCounter] = new Water(Double.parseDouble(waterTemp));


              /*
               * DO THE REST OF THE DATA FOR THE HOUR
               */
            do{
                dataLineString = m_input.readLine();
                if(dataLineString == null){
                    /*******************************************************
                     *the reader finds null while reading for the first time
                     *i.e. there is no data in the next line
                     *******************************************************/
                    //m_input.close();
                    prevLineString = null;
                    break;/*****Reached EOF******/
                }

                dataArray = dataLineString.split("\\t");

                year = dataArray[yearIndex];
                day  = dataArray[dayIndex];
                time = dataArray[timeIndex];

                /*****create a calendar for the time identification purpose*********/
                nextCalendar = new MyGregorianCalendar(year, yearInHeader, day, time);

                if(prevCalendar.get(GregorianCalendar.YEAR) == nextCalendar.get(GregorianCalendar.YEAR) &&
                    prevCalendar.get(GregorianCalendar.DAY_OF_YEAR) == nextCalendar.get(GregorianCalendar.DAY_OF_YEAR) &&
                    prevCalendar.get(GregorianCalendar.HOUR_OF_DAY) == nextCalendar.get(GregorianCalendar.HOUR_OF_DAY) )
                {
```

```
                    windDirection = dataArray[windDirectionIndex];
                    windSpeed     = dataArray[vectorWindSpeedIndex];

                    windArray[++localCounter] = new Wind(Double.parseDouble(windSpeed), Double.parseDouble(windDirection));

                    airTemp     = validateAndCollectData(airTempIndex,     dataArray);// dataArray[airTempIndex];
                         /* IF AIR TEMP IS BAD, GET FROM TABLE */
                    airPressure = validateAndCollectData(airPressureIndex, dataArray);// dataArray[airPressureIndex];
                    airRH       = validateAndCollectData(airRHIndex,       dataArray);// dataArray[airRHIndex];
                    airArray[localCounter] = new Air(Double.parseDouble(airTemp),
Double.parseDouble(airPressure),Double.parseDouble(airRH));


//                    System.out.println("YY:DD:TT  "+year+":"+day+":"+time);

                    waterTemp = validateAndCollectData(waterTempIndex, dataArray);
                         /* IF WATER TEMP IS BAD, GET FROM TABLE */
                    waterArray[localCounter] = new Water(Double.parseDouble(waterTemp));
                }

                else{

                    prevLineString = dataLineString;
                    break;
                }

        //prevLineString = null;
            //    ++cntForLoop;

            }while(true);

            //CD144 cd = null;
            if(m_averageCounter-1 <= localCounter){
                AverageWind avgWind = new AverageWind(m_averageCounter, windArray, stationInstance);
                AverageAir avgAir = new AverageAir(m_averageCounter, airArray);
                AverageWater avgWater = new AverageWater(m_averageCounter, waterArray);

                cd = new OutputDataCreater(stationInstance, prevCalendar, avgWind, avgAir, avgWater);
        avgWind = null;
                avgAir = null;
                avgWater = null;
            }
            else {
                cd = new OutputDataCreater(stationInstance, prevCalendar);
            }

            windArray = null;
            airArray = null;
            waterArray = null;
        }
        catch(IOException e){

//            reportError(e);
```

```
            System.out.println(e);

            cd = null;
            closeAllStreams();
        }
        catch(NullPointerException e){

            //System.out.println("End of File is reached: "+e.toString());
            cd = null;
            closeAllStreams();
        }

        return cd;
    }

//-------------------------------------------------------------------

    /**private methods begin****/

    private String validateAndCollectData(int index, String[] dataArray){
        if(index == -1 || dataArray[index].equals("") || dataArray[index].equals("-6999")){
            return "-6999";
        }
        else
            return dataArray[index];
    }


     private void closeAllStreams() {
         try {
             //m_errorStream.close();
             m_input.close();
         }

         catch(Exception e) {
             reportError(e);
         }
     }

//---------------------------------------------------------------


//-----------------------------------------------------------------------------
    private boolean isInArray(String[] array, String searchString){
         boolean found = false;

         for(int i = 0; i < array.length; i++){
            if(array[i].equalsIgnoreCase(searchString)){
                found = true;
                break;
            }
        }
        return found;
```

```
    }
//-------------------------------------------------------------------------
    private void reportError(Exception e) {
        try {
            //m_errorStream.println("Abnormal termination due to " + e.toString());
        }

        catch(Exception e2) {
            System.out.println();
        }
    }

//-------------------------------------------------------------------------

    /**
     * This method reports a user defined error string to the error stream
     * or to System.out if an exception occurs that prevents a write to the
     * error stream
     */
    private void reportError(String error) {
        try {
            //m_errorStream.println(error);
        }
        catch(Exception e2) {

        }
    }
//-------------------------------------------------------------------------
}/**end of this class***/
```

OutputDataCreater.java –

Holds all the data for a given set of input records, contains transformation functions for the WriterDriver file to display modified information.

```
package Batch2;

import java.util.*;
import java.sql.*;
import java.io.*;

/**
 * OutputDataCreater - Main data object.  Every OutputDataCreater
 *  object will write one line to the file.  All the calculations
 *  and variables are performed or stored by this object.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.ReaderDriver
 * @see Batch2.WriterDriver
 */
public class OutputDataCreater {


/*********inputs*****************************************************INPUTS*********************************************/
    public static final String CONFIG_FILE = Main.CONFIG_FILE;

    private static PrintWriter logger;
    static {
        try {  logger = new PrintWriter( new FileWriter( "log.txt", true ) ); }
        catch( Exception e ) { System.out.println("Error opening log.txt"); }
    }

    /** Current Particle Diameter for all ODC Calculations */
    public  static double diaPM            = PropertiesViewer.getDouble(CONFIG_FILE,"ValueForDiam");
    private static final double RhoPM      = PropertiesViewer.getDouble(CONFIG_FILE,"ValueForRhoPM");
    /** Current 1/Ra CAP for all ODC Calculations */
    public  static double CAP              = PropertiesViewer.getDouble(CONFIG_FILE,"ValueForCap");
    /** Z0 for offshore calculations */
    public  static final double Zo_OFFSHORE = PropertiesViewer.getDouble(CONFIG_FILE,"ValueForZo");

    /** Returns a string for the PM group based on the current diaPM */
    public static String getPMGroup()
        // used by main program to get info for the path //
    {
        if( diaPM == 1.0 || diaPM == 5.0 || diaPM == 15.0 )
            return "1_5_15";
        else if( diaPM == 2.0 || diaPM == 8.0 || diaPM == 20.0 )
            return "2_8_20";
        else if( diaPM == 2.5 || diaPM == 10.0 || diaPM == 25.0 )
            return "2.5_10_25";
        else
            return "other";
    }

    /** Returns the phosphorus concentration for the current diaPM */
```

```
    private double getPhosphorusConc()
    {
        if(diaPM <= 2.5) //microns
            return PFne;
        else if ( diaPM > 10 )
            return PCrs;
        else
            return PLrg;
    }

    /** Concentration for Phosphorus (PM2.5) */
    public static double PFne = 4;
    /** Concentration for Phosphorus (PMcrs) */
    public static double PCrs = 12;
    /** Concentration for Phosphorus (PMlrg) */
    public static double PLrg = 4;

    private Station m_station = null;
    private MyGregorianCalendar m_cal = null;
    private AverageWind m_avgWind = null;
    private AverageAir m_avgAir = null;
    private AverageWater m_avgWater = null;
    private static Connection m_con = null;
    static {
        System.out.println("Connecting to database...");
        createConnection();
    }

    protected boolean dataMissingFlag = true;

    double virtualPotentialTempAtSurfaceDegreeC = 0.0;
    double virtualPotentialAirWaterTempDiff = 0.0;

    private double uStar = 0.0;

    private double L_ForOnshore = 0.0;
    private double L_ForWDQI    = 0.0;

    private double VdForWDQI = 0.0;
    private double VdForOnshore = 0.0;

    private double Vg = 0;

    private double RaForWDQI = 0;
    private double RaForOnshore = 0;
    private ArrayList depositionRateValues = new ArrayList();
    private ArrayList depositionRateVariables = new ArrayList();


    /** public Constructor (connect to the database) */
    public OutputDataCreater() {
//        m_con = createConnection();
    }
```

```
    public OutputDataCreater(Station stationInst, MyGregorianCalendar cal) {

        m_station = stationInst;
        m_cal = cal;
//        m_con = createConnection();
        calculateVirtualPotentialTempAtSurface();
    }


    public OutputDataCreater(Station stationInst, MyGregorianCalendar calendar, AverageWind avgWind, AverageAir avgAir,
AverageWater avgWater) {

        m_station = stationInst;
        m_cal = calendar;

        m_avgWind = avgWind;
        m_avgAir = avgAir;
        m_avgWater = avgWater;

//        System.out.println(""+m_avgWind+" "+m_avgAir+" "+m_avgWater+" ");

        calculateVirtualPotentialTempAtSurface();
        Vg = calculateVg();


            // uStar is displayed for Ra for WDQI ! //
            //  the following method is the last method to modify uStar //
        RaForWDQI    = calculateDepositionVelocityWithAllObjectsForGas( calculateZo( false ) , false);       //uncapped
(more turbulent effect)
        RaForOnshore = calculateDepositionVelocityWithAllObjectsForGas( calculateZo( true  ) , true ); //uncapped

//        m_con = createConnection();

        VdForWDQI    = calculateDepositionVelocityWithAllObjectsForPM( applyOneOverCap(RaForWDQI) );
        VdForOnshore = calculateDepositionVelocityWithAllObjectsForPM( applyOneOverCap(RaForOnshore) );

        calcDepositionRate();


//        closeConnection();
    }

//=============================================================================
    /**********public methods begin**************/

    public Station getStation(){  return m_station; }
    public MyGregorianCalendar getCalendar(){  return m_cal; }
    public AverageWind getAvgWind(){  return m_avgWind;  }
    public AverageAir getAvgAir(){  return m_avgAir;   }
    public AverageWater getAvgWater(){  return m_avgWater;   }


    private double z0wdqi;
```

```
    private double z0onshore;

    public double getZ0_ForWDQI(){  return z0wdqi;  }
    public double getZ0_ForOnshore(){  return z0onshore;  }

    public double getuStar(){  return uStar;  }
    public double getL_ForWDQI(){  return L_ForWDQI;  }
    public double getL_ForOnshore(){  return L_ForOnshore;  }
    public double getVdForWDQI(){ return VdForWDQI;  }
    public double getVdForOnshore(){  return VdForOnshore;  }
    public double getVg(){  return Vg;  }
    public double getDiaPM(){  return diaPM; }
    public double getRhoPM(){  return RhoPM; }
    public double getVdPMminusVgForWDQI(){  return VdForWDQI - Vg; }
    public double getVdPMminusVgForOnshore(){ return VdForOnshore - Vg; }

    public String getOneOverRaWithZoWDQI(){
        if(RaForWDQI == 0)
            return "";
        if(RaForWDQI < (double)1/(double)CAP)
            return ""+CAP;
        return ""+(1 / RaForWDQI);
    }
    public String getOneOverRaWithZoOnshore(){
        if(RaForOnshore == 0)
            return "";
        if(RaForOnshore < (double)1/(double)CAP)
            return ""+CAP;
        return ""+(1 / RaForOnshore);
    }

    public static final boolean USE_CAP = PropertiesViewer.getBoolean(Main.CONFIG_FILE,"UseOneOverRaCap");
    public double applyOneOverCap( double doThisRa )
    {
        if( USE_CAP && doThisRa <= 1/CAP )
            return 1/CAP;
        else
            return doThisRa;
    }

    public double getVirtualPotentialAirWaterTempDiff(){
        virtualPotentialAirWaterTempDiff = m_avgAir.getVirtualPotentialTempDegreeK() -
getVirtualPotentialTempAtSurfaceDegreeK();
        return virtualPotentialAirWaterTempDiff;
    }

    public double getBulkRichardsonNo(){
        double g = 9.8;
        double airTemp = m_avgAir.getVirtualPotentialTempDegreeK();
        double sensorHeight = m_station.getSensorHeight();
        double windSpeed = m_avgWind.getWindSpeedInMetersPerSecond();
        double tempDifference = getVirtualPotentialAirWaterTempDiff();
        if(airTemp == 0 || windSpeed == 0){
//            System.out.println("dMF@getBulkRichardsonNo()");
```

```
            dataMissingFlag = true;
            return 0;
        }

        double RiB = (g * tempDifference * sensorHeight) / (airTemp * windSpeed * windSpeed);
        return RiB;
    }

    public double getVirtualPotentialTempAtSurfaceDegreeK(){
        return 273.16 + virtualPotentialTempAtSurfaceDegreeC;
    }

    public boolean isDataSet(){
        if(dataMissingFlag == true){
            return false;
        }
        return true;
    }

    public String getPMSizeRange(){
        if(diaPM <= 2.5) //microns
            return "PM2.5";
        else if ( diaPM > 10 )
            return "PMlrg";
        else
            return "PMcrs";

    }

    public ArrayList getDepositionRateVariables(){
        return depositionRateVariables;
    }

    public ArrayList getDepositionRateValues(){
        return depositionRateValues;
    }
//=====================================================================
    /** Close this classes Active connection to the db */
    public static void closeConnection(){
        try {
            if(m_con != null)
            m_con.close();
            m_con = null;
        }catch(SQLException e) {}
    }

    //---------------------------------------------------------------------
    private Connection createConnection(){
        if( m_con == null )
        {
          Connection con = null;
            try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
```

```
                con = DriverManager.getConnection("jdbc:mysql://RD-20021555:3306/ltads_2" +
                    "?socketFactory=com.mysql.jdbc.NamedPipeSocketFactory", "root", "xyxyxy");

                if(con.isClosed()){
                    System.out.println("Problem with the Database...");
                    System.out.println("exiting...");
                    System.exit(1);
                }
                return con;
            }
            catch(Exception e) {
                System.err.println("Exception: " + e);
            }
        }
        return null;
    }
//-----------------------------------------------------------------------------

    private static final double fNO3 = PropertiesViewer.getDouble(Main.CONFIG_FILE,"DepRatePMCrsWeight");
    private static final int drtablemod = PropertiesViewer.getInt( Main.CONFIG_FILE,"UseModifiedVariableTables" );
    public  static String calcDepositionRateTableName;  // modified by Main


    public static final String LOW_END_TABLE_NUM;
    static {
        if( PropertiesViewer.getBoolean(Main.CONFIG_FILE,"Force206020Tables") )
            LOW_END_TABLE_NUM = "206020";
        else
            LOW_END_TABLE_NUM = "8020";
    }

    public static String getTableName( double dpm )
    {
        if( dpm == 2.5 )  // pretend that 2.5 = 3
            dpm = 3;
        switch( (int)dpm )
        {
            case 1: case 5: case 15:
                if( CAP == 3.0 )
                    return "seasonalavgp40_" + LOW_END_TABLE_NUM + getModifier(drtablemod);
                else
                    return "seasonalavgp40_206020" + getModifier(drtablemod);
            case 2: case 8: case 20:
                return "seasonalavgp40_206020"+ getModifier(drtablemod);
            case 3: case 10: case 25:
                return "seasonalavgp40_206020"+ getModifier(drtablemod);
            default:
                return "seasonalavgp40_206020"+ getModifier(drtablemod);
        }
    }
```

```
    private static final String[] varTypes = { "HNO3", "NH3", "Mass", "NH4", "NO3", "P" };
    private static final String[] staTypes = { "gas",  "gas", "PM",   "PM",  "PM",  "PM" };

    private void calcDepositionRate( ){
    //    System.out.println(calcDepositionRateTableName);
        ResultSet rs = null;
        ResultSet rs2 = null;
        double weightedAvg =0;



        String tableName = calcDepositionRateTableName;

        String colName = "AvgConc";
        String state = "";

        String ltadsID = m_station.getCorresStation();
        String season  = m_cal.getSeason();
        double tempdiurnalMultiplier;
        String gasOrPM;

        for( int j = 0; j < varTypes.length ; j++ )
        {
            gasOrPM = staTypes[j];

            if(gasOrPM.equalsIgnoreCase("gas"))  {
                  /** GAS!!! **/
                state = "gas";
                weightedAvg =  calculateWeightedAvg( getOneOverRaWithZoWDQI(), getOneOverRaWithZoOnshore());
                tempdiurnalMultiplier = 0;
            } else {
                state = getPMSizeRange(); // = PM2.5, PMcrs, PMlrg
                weightedAvg =  calculateWeightedAvg(""+VdForWDQI, ""+VdForOnshore);
                tempdiurnalMultiplier = calculateSeasonalHrlyAvgConcPM("PM2.5");
            }


            String sql = "SELECT "+colName+" FROM " +tableName+ " WHERE LtadsID = '"
                        + ltadsID +  "' and Season = '" + season+"' and State = '"
                        + state + "' and Variable = '" + varTypes[j] + "'";
            int i = 0;

            try{
//                  while(rs.next())
//                  {

                if( !varTypes[j].equals("P") )
                {
                    rs = returnResultSetFromDB(sql);
                    rs.next(); //
                }

                    double seasonalAvgConc = 0;
                    double diurnalMultiplier = 0;
```

```
                    if(gasOrPM.equalsIgnoreCase("gas")){
                        seasonalAvgConc = rs.getDouble(1);
                        diurnalMultiplier = calculateSeasonalHrlyAvgConcGas(varTypes[j]);
                    }
                    else{   // We're dealing with PM
                        if( varTypes[j].equalsIgnoreCase("NH4") || varTypes[j].equalsIgnoreCase("NO3") )
                        {
                            sql =  "SELECT "+colName+" FROM " +tableName+ " WHERE LtadsID = '"+ltadsID+
                                    "' and Season =  '"+season+"' and ( State = 'PMcrs' or State = 'PMlrg' ) and" +
                                    " Variable = '"+varTypes[j]+"'";

                                /**
                                 *  QUERY DATABASE  : seasonalavgp40_8020 | seasonalavgp40_206020
                                 ***/
                            rs2 = returnResultSetFromDB(sql);
                            int counter = 0;
                            while(rs2.next()){
                                seasonalAvgConc += rs2.getDouble(1);
                                counter++;
                            }
                            rs2 = null;

                            if(state.equals("PMcrs")){
                                seasonalAvgConc *= fNO3;       // * 0.5
                            }
                            else{ // state equals PMlrg
                                seasonalAvgConc *= (1 - fNO3); // * 0.5
                            }
                            diurnalMultiplier = calculateSeasonalHrlyAvgConcPM( state );  //NH4 & NO3 get ratio from BAM-
PM2.5
                        }
                        else // Mass or P
                        {
                            if( !(varTypes[j].charAt(0) == 'P') )
                            {
                                seasonalAvgConc = rs.getDouble(1);
                            }
                            else
                            {
                                seasonalAvgConc = getPhosphorusConc();
                            }
                            diurnalMultiplier = calculateSeasonalHrlyAvgConcPM( state );
                        }

                    }
                    depositionRateValues.add(Double.valueOf(""+weightedAvg * seasonalAvgConc * diurnalMultiplier));
                    depositionRateVariables.add(varTypes[j]);
//                }
            }
            catch(SQLException e){
                System.err.println("Exception: ");
                e.printStackTrace();
            }
```

```
        }
    }
    //-----------------------------------------------------------------------
    private ResultSet returnResultSetFromDB(String sql){
        try {
            Statement st = null;
            st = m_con.createStatement();

            return st.executeQuery(sql);
        }
        catch(SQLException e) {

            System.err.println("Exception: " + e.getMessage());
            try{
                if(m_con != null)
                    m_con.close();
            }
            catch(SQLException e1){
                System.err.println("Exception:"+ e.getMessage());
            }
        }
        catch( Exception e )
        {
            m_con = createConnection();
            return returnResultSetFromDB(sql);
        }
        return null;
    }


    /*-----------------------------------------------------------------------
     *  The following two functins use data contained in the concentration
     *   database tables
     */


    private double calculateSeasonalHrlyAvgConcGas(String variable){
        String tableName = "gasseasonhrlyconcratio";  // HERE
        String colName = "ConcRatio";
        ResultSet rs = null;
        String sql =  "SELECT "+colName+" FROM " +tableName+ " WHERE Season = '"+m_cal.getSeason()+
                      "' and Variable = '"+variable+"' and StrtHr = '"+m_cal.get(Calendar.HOUR_OF_DAY)+"'";


        rs = returnResultSetFromDB(sql);
        try{
            rs.first();
            return rs.getDouble(1);
        }
        catch(SQLException e){
            System.err.println("Exception: "+e.getMessage());
        }
        return 0;
```

```
    }
    //-----------------------------------------------------------------------

    private static final boolean newConcTables =
PropertiesViewer.getBoolean(CONFIG_FILE,"UseModifiedLFandTBConcentrations");
    private double calculateSeasonalHrlyAvgConcPM(String state) // state is PM2.5, PMlrg, PMcrs
      // calculate the Multiplier for PM based data from
      //  the hourly BAM concentrations
    {
        ResultSet rs = null;

/*        String tableName = "seasonalhrlyavgconc"; // AND HERE
        if( newConcTables )  // newer concentration data (with Bliss and SLSW TSP)
            tableName+="_mod"; */

        String tableName = "bam_conc_pmfine";

        String ratioField = "ConcRatio";
        String ltadsID = m_station.getCorresStation();
//        int year = m_cal.get(Calendar.YEAR);
        int year = 2003;
        int hour = m_cal.get(Calendar.HOUR_OF_DAY);
        String season = m_cal.getSeason();

        if (! state.equals("PM2.5"))  // for PMcrs,PMlrg
        {
            if( ltadsID.equals( "SW" ) )
            {
                ltadsID = "SLSW";    // For Sandy Way, Mass, dPM > 2.5,
                state = "TSP";  //  use the Sandy Way/SOLA TSP For ratios.
            }
            else  // otherwise, combine PMcrs/PMlrg
            {
                tableName = "bam_conc_pmcrslrg";
            }
        }

        String sql =  "SELECT "+ratioField+" FROM "+tableName+" WHERE LtadsID = '"+ltadsID+
                      "' and Year = '"+year+"' and Season =  '"+season+
                      "' and State = '["+state+"]' and StrtHr='"+hour+"'";
        try
        {
            rs = returnResultSetFromDB(sql);
            rs.first();
            return rs.getDouble(ratioField);
        } catch ( SQLException e )
        {
            System.out.println("SQL ERROR!");
            e.printStackTrace();
            return 0;
        }


/*
```

```
        //*************retrieve from the table provided by BAM data*****************
        String sql =  "SELECT AvgConc FROM "+tableName+" WHERE LtadsID = '"+ltadsID+
                      "' and Year = '"+year+"' and Season =  '"+season+"' and (State = '["+stateDenom+"]')";


        //** DENOMINATOR CALC: Seasonal Hourly Average *
        rs = returnResultSetFromDB(sql);  // return 24/48 data points
        int counter = 0,
            counter2 = 0;
        try{
            while(rs.next()) {
                double d = rs.getDouble("AvgConc");  // get ALL of this season's hourly conc's //
                if(d < 0){
                    d = 0; // Force Negative Concentrations from PM seasonal hourly to 0
                }
                denominator += d; // add the positive ones!
                counter++;        // increment a counter!
            }

            // Counter Should equal either 24 or 48; 48 -> PMcrs|PMlrg, 24 -> PM2.5

               // NO (or BAD) DATA //
            if( counter == 0 || denominator ==0 ){
                return 0;
            }
            else{
                if((! state.equals("PM2.5"))  && !ltadsID.equals("SLSW") ){
                    counter = counter / 2; // divide count by two to do PMcrs+PMlrg
                }
                if( counter != 24 )
                {
                    System.out.println("BAM counter is not 24!");
                    while( true )
                    { }
                }
                denominator /= counter; // always divide by 24
            }

            /*********now calculate numerator value***********
            sql =  "SELECT AvgConc FROM "+tableName+" WHERE LtadsID = '"+ltadsID+
                          "' and Year = '"+year+"' and Season =  '"+season+"' and "+
                          "StrtHr = '" + hour + "' and (State = '["+stateDenom+"]')";

            //** NUMERATOR CALC: Seasonal Hour's Concentration *
            rs = returnResultSetFromDB(sql);
//            rs.first();
            while(rs.next()) {
                double d = rs.getDouble("AvgConc");  //current concentration (per hour per type ('PMlrg' or
'PMcrs'|'PM2.5') )
                if(d < 0){
                    d = 0; // Force Negative Concentrations from PM seasonal hourly to 0
                }
                numerator+=d;
                counter2++;
```

```
            }

            if((! state.equals("PM2.5"))  && !ltadsID.equals("SLSW") ){
                counter2 /= 2;
            }
            if( counter2 != 1 )
            {
                System.out.println("BAM counter2 is not 1!");
                while( true )
                { }
            }
            numerator /= counter2;  // divide by 1

        }
        catch(SQLException e){
            System.err.println("Exception: "+e.getMessage());
        }
        if (denominator == 0 )
            return 0;

        return numerator / denominator;
 *
 */



    }

    //-----------------------------------------------------------------------
    private double calculateWeightedAvg(String s1, String s2){
        if(s1.equals("") || s2.equals("")){
            return 0;
        }

        double a = 0.2;
        double b = 0.8;
        double v1 = 0;
        double v2 = 0;


        v1 = Double.parseDouble(s1);
        v2 = Double.parseDouble(s2);

        return (a*v1 + b*v2)/(a+b);
    }

    /*
     *   New Functions
     */

    private double calculateUStar( double uZ, double k, double z, double Zo, double L )
    {
        if( L > 0 ) {
```

```
            return (uZ *k) / (Math.log((z-Zo)/Zo) + 4.7*z/L );
        } else if( L < 0 ) {
            double numerator = (Math.sqrt(1+16*z/Math.abs(L))-1)*(Math.sqrt(1+16*Zo/Math.abs(L))+1);
            double denom     = (Math.sqrt(1+16*z/Math.abs(L))+1)*(Math.sqrt(1+16*Zo/Math.abs(L))-1);
            return (uZ * k) / Math.log( numerator / denom );
        } else {
            dataMissingFlag = true;
//            System.out.println("dMF@calculateUStar()");
            return 0;
        }
    }

    private double calculateU10( double Ustar, double k, double Zo, double z, double L)
    {
        if( L > 0 ) {
            return (Ustar/k) * (Math.log((10-Zo)/Zo) + 4.7*z/L );
        } else if( L < 0 ) {
            double numerator = (Math.sqrt(1+16*10/Math.abs(L))-1)*(Math.sqrt(1+16*Zo/Math.abs(L))+1);
            double denom     = (Math.sqrt(1+16*10/Math.abs(L))+1)*(Math.sqrt(1+16*Zo/Math.abs(L))-1);
            return (Ustar / k) * Math.log( numerator / denom );
        } else {
            dataMissingFlag = true;
//            System.out.println("dMF@calculateU10()");
            return 0;
        }
    }

    private double calculateCuN( double u10 )
    {
        return (.75+(.067*u10))/1000;
    }

    private double calculateL( double Tair, double Twater, double CuN, double u10, double E2 )
    {
        return (Tair+273.16)*Math.pow(CuN,1.5)*u10*u10/(E2*(Tair-Twater));
    }

    private double calculateRa( double z, double L, double Zo, double k, double uStar )
    {
        if( L > 0 ) {
            return (1/(k*uStar))*(Math.log((z-Zo)/Zo) + 4.7*z/L )/100;//CONVERTING TO S/CM
        }
        else if( L < 0 ) {
            double numerator = (Math.sqrt(1+16*z/Math.abs(L))-1)*(Math.sqrt(1+16*Zo/Math.abs(L))+1);
            double denom     = (Math.sqrt(1+16*z/Math.abs(L))+1)*(Math.sqrt(1+16*Zo/Math.abs(L))-1);
            return (1/(k*uStar))*Math.log(numerator/denom)/100;//CONVERTING TO S/CM
        }
        else {
//            System.out.println("dMF@calculateRa()");
            dataMissingFlag = true;
            return 0;
        }
    }
```

```
private double calculateZo( boolean onshore, double u10 )
{
    double windSpeedMPS = u10;
    int wdqi = m_avgWind.getWDQI();
    if(wdqi == -1 || onshore )
        return 0.000002 * Math.pow(windSpeedMPS, 2.5);
    if(wdqi  == 1)
        return Zo_OFFSHORE;
    if(wdqi == 0)
        return (Zo_OFFSHORE + 0.000002 * Math.pow(windSpeedMPS, 2.5))/2;
    return 0;
}




/*
 *  End new functions
 */

//-----------------------------------------------------------------------------

private double u10_saved = 0.0;
public double getU10() { return u10_saved; }
public static final double RA_SENSITIVITY = PropertiesViewer.getDouble( CONFIG_FILE, "RaSensitivity");
private static final int MAX_ITERATIONS = PropertiesViewer.getInt( CONFIG_FILE, "MaxIterations");
private static final boolean LOG_LIMIT_EXCEEDED = PropertiesViewer.getBoolean( CONFIG_FILE, "LogLimitExceeded");

private double calculateDepositionVelocityWithAllObjectsForGas(double Zo, boolean onshore )
/*
    Calculates Ra for Gas (not PM)
 */
{
    final double uZ     = m_avgWind.getWindSpeedInMetersPerSecond() ;//* 100; //converted to cm/s
    double Ra = 0;
    if(uZ != 0){

        final double E2= 0.0051;
        final double k = 0.4; // Von Karmen constant (0.35-0.4)
        final double Tair   = m_avgAir.getAvgAirTempDegreeC();
        final double Twater = m_avgWater.getAvgTempDegreeC();
        final double z      = m_station.getSensorHeight(); // z in meters
        int i;

        double  CuN,     //
                Z0 = Zo, //  intitial Zo
                L = Tair-Twater,       //
                Ustar,   //
                U10;     //

        Ustar = calculateUStar( uZ, k, z,  Z0, L );  //uses initial Zo
        U10   = calculateU10( Ustar, k, Z0, z, L );  //uses initial Zo
```

```
//          U10   = calculateU10( Ustar, k, Zo, z, L );  //uses initial Zo
            Z0    = calculateZo( onshore, U10);          // Z0 is the new Zo
            CuN   = calculateCuN( U10 );
            L     = calculateL(Tair, Twater, CuN, U10, E2 );
            Ra    = calculateRa(z, L,Z0, k, Ustar );

            // Initialize Variables: Z0, L //

            // Run calculation pump //
            double old_Ra = Ra + 100;
//            for( i = 0; i < 2 ; i++ )
//            for( i = 0; Math.abs(old_Ra - Ra) >= RA_SENSITIVITY; i++ )
            for( i = 0; Math.abs(old_Ra - Ra) >= RA_SENSITIVITY && i < MAX_ITERATIONS; i++ )
            {
                Ustar = calculateUStar( uZ, k, z,  Z0, L );
                U10   = calculateU10( Ustar, k, Z0, z, L );
//              U10   = calculateU10( Ustar, k, Zo, z, L );
                Z0    = calculateZo( onshore, U10);
                CuN   = calculateCuN( U10 );
                L     = calculateL(Tair, Twater, CuN, U10, E2 );
                old_Ra = Ra;
                Ra    = calculateRa(z, L,Z0, k, Ustar );
//              System.out.println("("+i+") Value of Ra: " + Ra + " versus Ustar:" + (uZ/(Ustar*Ustar)) + " OldRa="+old_Ra
);
            }
            if( i >= MAX_ITERATIONS && LOG_LIMIT_EXCEEDED )
            {
                logger.println( ""+i+" iterations (exceeded) for " +m_station.getName()+"_"+m_station.getCorresStation() );
                logger.println( " at "+ m_cal.getTime() );
                logger.println( " ATEMP="+m_avgAir.getAvgAirTempDegreeC() );
                logger.println( " WTEMP="+m_avgWater.getAvgTempDegreeC() );
                logger.println( " OldRa="+old_Ra + " Ra="+Ra  );
                logger.flush();
            }

            uStar = Ustar;      // save to object globals...
            u10_saved = U10;
            if( onshore )
            {
                z0onshore = Z0;
                L_ForOnshore = L;
            }
            else
            {
                z0wdqi = Z0;
                L_ForWDQI = L;
            }

            return uZ/(Ustar*Ustar)/100;//CONVERTING TO S/CM
        }
//        dataMissingFlag = true;
//        System.out.println("dMF@calculateDepositionVelocityWithAllObjectsForGas()");
        return 0;
    }
```

```
private static final boolean useU10ForCounterHack = PropertiesViewer.getBoolean(CONFIG_FILE,"UseU10ForCounterHack");

private static final boolean doNewPMCalc = PropertiesViewer.getBoolean(CONFIG_FILE,"UseNewVdPMCalculation");
private double calculateDepositionVelocityWithAllObjectsForPM(double Ra){
        //Now start calculations needed specifically for particle deposition velocity: Rb, Vg
        if(uStar != 0){
            double a1 = 1.257;
            double a2 = 0.4;
            double a3 = 0.000055;
            double x2 = 0.0000065;
            double Vdphoresis =0;            ///// 0.01 ; effect of evaporating surface

            double Tair = m_avgAir.getAvgAirTempDegreeC();


            double Scf = 1 + 2*x2*(a1 + a2* Math.exp(-a3*diaPM/x2))/(diaPM*0.0001);// Cunningham slip correction factor
            double Db = 8.09*0.0000000001*(Tair+273.16)*Scf/diaPM ;// Brownian diffusivity (cm/s);
            double Visc_air= 0.15; // viscosity of air (~0.15 cm2/s)
            double Sc = Visc_air / Db; //Schmidt number in cm;


                    // This uses the uStar value from the most recent Vd calc for Gas
            double St = (Vg/981)*uStar*uStar/Visc_air; //Stokes number ; acceleration of gravity = 981 cm/s/s
            double Rd = (1/uStar)/(Math.pow(Sc,-2/3) + Math.pow(10,-3/St));

            if( doNewPMCalc )
                return Vg/(1-Math.exp(-Vg*(Ra+Rd+0)));
            else
                return  1/(Ra + Rd + Ra*Rd*Vg) +Vg + Vdphoresis;

        }
        dataMissingFlag = true;
        return 0;
}

/*************this method is called by  getDepositionVelocityFromStationHeight() method*******************/

private double calculateZo( boolean onshore ){
    // calculates the initial Zo for depvel calculations

    double windSpeedMPS = m_avgWind.getWindSpeedInMetersPerSecond();
    int wdqi = m_avgWind.getWDQI();
    if(wdqi == -1 || onshore )
        return 0.000002 * Math.pow(windSpeedMPS, 2.5);
    if(wdqi  == 1)
        return Zo_OFFSHORE;
    if(wdqi == 0)
        return (Zo_OFFSHORE + 0.000002 * Math.pow(windSpeedMPS, 2.5))/2;
    return 0;
}

/************************gravitational settling velocity****************************************/
```

```
    private double calculateVg(){ //settling velocity calculation
            double a1 = 1.257;
            double a2 = 0.4;
            double a3 = 0.000055;
            double x2 = 0.0000065;
            double c2 = 0.00000001; //conversion cm2/micrometers2 : conversion multiplier from micrometer2 to cm2
            double Mu = 0.000181 ; //absolute viscosity of air g/cm/s

            double pressure = m_avgAir.getAvgAirPressure(); // pressure in miliBars
            double Tair = m_avgAir.getAvgAirTempDegreeC();

            double Scf = 1 + 2*x2*(a1 + a2* Math.exp(-a3*diaPM/x2))/(diaPM*0.0001);// Cunningham slip correction factor

            //Rho_pm = input ; g/cm3 (~1-3)
            double Rho_air = 0.0012*(pressure/1000)*273.16/(Tair+273.16); //g/cm3 approximate
            //diaPM = input; particle diameter in micrometers

            return ((RhoPM - Rho_air)*981*diaPM*diaPM*c2)*Scf/(18*Mu) ;// gravitational settling velocity
    }

    private double calculateVirtualPotentialTempAtSurface(){
        if( m_avgWater == null || m_avgAir == null || !m_avgWater.isDataSet() || !m_avgAir.isDataSet()) {
/*            System.out.println("dMF@calculateVirtualPotentialTempAtSurface():missingdata");
            if( m_avgWater != null && m_avgAir != null  )
                System.out.println("  avgWater.isDataSet()="+m_avgWater.isDataSet()+"
avgAir.isDataSet()="+m_avgAir.isDataSet());
            else
                System.out.println("  avgWater==null:"+(m_avgWater==null)+"  avgAir==null:"+(m_avgAir==null));*/
            dataMissingFlag = true;
            return 0;
        }

        double waterTemp = m_avgWater.getAvgTempDegreeC();
        double airPressure = m_avgAir.getAvgAirPressure();

        double e = 6.107 * Math.pow(10,(7.5*waterTemp/(237+waterTemp)));
        double w = 0.623 * (e / airPressure);
        double Tv0 = waterTemp * (1 + 0.61 * w);

        virtualPotentialTempAtSurfaceDegreeC = Tv0 * Math.pow((1000 / airPressure), 0.286);
        dataMissingFlag = false;
        return virtualPotentialTempAtSurfaceDegreeC;
    }

    private static String getModifier( int i )    {
        if( i==1 ) return "_mod";
        else if( i > 1 ) return "_mod"+i;
        else return "";
    }


}
```

WriterDriver.java –

Organizes data from OutputDataCreater, appending it to a .dpR output file.

```
package Batch2;

import java.util.*;
import java.io.*;
import java.lang.*;
import java.math.BigDecimal;
import java.util.Date;

/**
 * WriterDriver - A Class that uses an OutputDataCreater object
 *  to display a row of data to an output file
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @see Batch2.OutputDataCreater
 */
 public class WriterDriver {
    /***private variables for the switch case statement****/
    private static int YEAR = 0;
    private static int MONTH = 1;  // Month index
    private static int DAY = 2;   // day index
    private static int HOUR = 3;
    private static int WIND_DIRECTION_IN_DEGREES = 5;
    private static int WIND_SPEED = 6;
    private static int WDQI = 9;
    private static int A_PRESS = 10;
    private static int RH = 11;
    private static int A_TEMP = 12;
    private static int W_TEMP = 13;
    private static int TEMP_DIFF = 14;
    private static int MIXING_RATIO = 15;
    private static int DEPOSITION_VELOCITY = 21;

        /***private variable for looping****/
    private int m_LOOPING_COUNTER_FOR_HOUR =0;
    private OutputDataCreater m_opd = null;

    /****create the private variables for the CD144 format*****/
    private static String m_surfaceStationName = "";
    private static String m_surfaceStationNumber = "";
    private static String m_year = "";
    private static String m_month ="";
    private static String m_day ="";
    private static String m_hour ="";
    private String m_windDirectionInDegrees ="";
    private String m_windSpeed ="";


    private String m_WDQI = "";
    private String m_airPressure = "";
    private String m_airRH = "";
    private String m_airTemp = "";
    private String m_waterTemp = "";
```

```
    private String m_tempDiff = "";
    private String m_mixingRatio = "";

    private String m_depositionVelocityFromBulkTransferCOef = "";
    private String uStar = "";

    private String VdPMminusVgWDQI = "";
    private String VdPMminusVgOnshore = "";
    private String VdWDQI = "";
    private String VdOnshore = "";
    private String Vg = "";

    private String m_airVpTemp = "";
    private String m_waterVpTemp = "";
    private String m_airWaterVpTempDiff = "";
    private String m_bulkRichardsonNo = "";
   /****create private variables for the summationU and summationV******/
    private double summationU = 0.0;
    private double summationV = 0.0;


    MyGregorianCalendar cal = null;
    AverageWind avgWind = null;
    AverageAir avgAir = null;
    AverageWater avgWater = null;
    private static boolean oneTimeWriterFlag = true;


     /** Output Stream, accessible by external objects */
    public static PrintWriter m_cd144Stream = null;
    /** Displays Header Fields when true.  Auto-clears. */
    /** Determines whether or not to display "alternate" data
     *   such as 10/L, uStar, z0-Onshore, etc...
     */
    public static final boolean tenOverL_Hack = PropertiesViewer.getBoolean( Main.CONFIG_FILE,
"SubstituteVTempsForTenOverL" );
    /** String containing the Header Row */
    public static final String HEADER_FIELDS;
    static {
        if( tenOverL_Hack )
            HEADER_FIELDS = "YYYY\tMM\tDD\tHH\twS\twD\twDQI\tuStar\tDepVelBlkC\t1/RaWDQI\t1/RaOnshore\t" +
            "Z0_WDQI\tZ0_Onshore\tU10\tVdWDQI\tVdOnshore\tA_Pres\tRH\tA_Temp\tW_Temp\tTempDiff\t10/L-WDQI\t10/L-
Onshore\tVPTempDiff\tRiB\tMxR\t" +
            "DepRateHNO3\tDepRateNH3\tDepRateMass\tDepRateNH4\tDepRateNO3\tDepRateP";
        else
            HEADER_FIELDS = "YYYY\tMM\tDD\tHH\twS\twD\twDQI\tuStar\tDepVelBlkC\t1/RaWDQI\t1/RaOnshore\t" +
            "VdPM-Vg_WDQI\tVdPM-
Vg_Onshore\tVg\tVdWDQI\tVdOnshore\tA_Pres\tRH\tA_Temp\tW_Temp\tTempDiff\tA_VpTemp\tW_VpTemp\tVPTempDiff\tRiB\tMxR\t" +
            "DepRateHNO3\tDepRateNH3\tDepRateMass\tDepRateNH4\tDepRateNO3\tDepRateP";
    }

/** Default constructor
 */
    public WriterDriver() {
```

```
        m_year = "00";
        m_month = "00";
        m_day = "00";
        m_hour = "00";

        m_windDirectionInDegrees = "00";
        m_windSpeed = "00";

    }

/** Initializes the internal data structures with references to the correct data
 * @param opd The OutputDataCreator object containing data for the current hour
 */
    public WriterDriver( OutputDataCreater opd ){
        m_opd = opd;
    /****create variables for the calendar and wind array******/
        cal      = m_opd.getCalendar();
        avgWind  = m_opd.getAvgWind();
        avgAir   = m_opd.getAvgAir();
        avgWater = m_opd.getAvgWater();

        convertAllVarToOutputFormat();


    }

    /*********public methods begin*************/
    //-----------------------------------------------------------------------


     /** old output file */
    private static String ifName = "";

    public void writeToFile( String outputFileName ){
        try{
              // New file encountered //
            if( !outputFileName.equals(ifName) )
            {
                if( m_cd144Stream != null )
                    m_cd144Stream.close();
                ifName = outputFileName;
                m_cd144Stream = new PrintWriter(new BufferedWriter(new FileWriter(outputFileName, true)));
                oneTimeWriterFlag = true;
            }

            if(oneTimeWriterFlag == true){
                writeHeader(outputFileName);
                oneTimeWriterFlag = false;
            }

            m_cd144Stream.print(
                m_year +"\t"+
                m_month +"\t"+
```

```
                m_day +"\t"+
                m_hour +"\t"+
                m_windSpeed+"\t"+
                m_windDirectionInDegrees+"\t"+
                m_wDQI+"\t"+
                uStar +"\t"+
                m_depositionVelocityFromBulkTransferCOef +"\t"+
                m_opd.getOneOverRaWithZoWDQI()+"\t"+
                m_opd.getOneOverRaWithZoOnshore()+"\t"+
                VdPMminusVgWDQI+"\t"+     // HACK - Z0
                VdPMminusVgOnshore+"\t"+  // HACK - Z0
                Vg +"\t"+                 // HACK - u10
                VdWDQI+"\t"+
                VdOnshore+"\t"+
                m_airPressure+"\t"+
                m_airRH+"\t"+
                m_airTemp+"\t"+
                m_waterTemp+"\t"+
                m_tempDiff+"\t"+
                m_airVpTemp+"\t"+    //tenOverL_Hack
                m_waterVpTemp+"\t"+  //tenOverL_Hack
                m_airWaterVpTempDiff+"\t"+  // replace with
                m_bulkRichardsonNo+"\t"+
                m_mixingRatio
            );

            if(m_opd != null &&  m_opd.isDataSet()){
                printValues();
            }
            m_cd144Stream.println();
        }catch(FileNotFoundException e){

            System.out.println("File Not Found Exception Occured: "+e.toString());
        }catch(IOException e){
            System.out.println("Problem with output file: "+e.toString());
        }
        finally{
/*          m_cd144Stream.flush();
            m_cd144Stream.close();
            m_cd144Stream = null;  */
        }
    }


    /** Private method begin */
    private void writeHeader(String outputFileName)
    {
        Station st = m_opd.getStation();
        m_cd144Stream.println("**************************************************");
        m_cd144Stream.println("* Output Showing Wind Direction, Wind Speed,    *");
        m_cd144Stream.println("* Wind Quality Index, Air and Water Temp etc.   *");
        m_cd144Stream.println("**************************************************\n");
        m_cd144Stream.println("File created on :"+ new Date());
```

```
        m_cd144Stream.println("YYYY: Year\tMM: Month\tDD: Day\tHH: Hour");
        m_cd144Stream.println("WS: WindSpeed (meters/sec)\tWD: WindDirection (degrees)");
        m_cd144Stream.println("WDQI: Wind Dir Quality Index(offshore[+1]/onshore[-1]/sieshore[0]/undefined[2])");
        m_cd144Stream.println("DepVelBlkC (centimeter/sec)");
        m_cd144Stream.println("A_Pres: Air Pressure (miliBars)\tRH: Relative Humidity(Fraction)");
        m_cd144Stream.println("A_Temp: Air Temp (degree celcius)");
        m_cd144Stream.println("W_Temp: Water Temp (degree celcius)");
        m_cd144Stream.println("TempDiff: A_Temp - W_Temp (degree celcius)");
        m_cd144Stream.println("A_VpTemp: Virtual Potential Air Temp (degree kelvin)");
        m_cd144Stream.println("W_VpTemp: Virtual Potential Water Temp (degree kelvin)");
        m_cd144Stream.println("VPTempDiff: Virtual Potential Air-Water Temp Difference (degree kelvin)");
        m_cd144Stream.println("RiB: Bulk Richerdson #");
        m_cd144Stream.println("MxR: Mixing Ratio");
        m_cd144Stream.println("DepRateHNO3: Deposition Rate of HNO3");
        m_cd144Stream.println("DepRateNH3: Deposition Rate of NH3");
        m_cd144Stream.println("DepRateP: Deposition Rate of Phosporous");
        m_cd144Stream.println("DepRateNH4: Deposition Rate of NH4");
        m_cd144Stream.println("DepRateNO3: Deposition Rate of NO3");
        m_cd144Stream.println("DepRateMass: Deposition Rate of Mass\n");
        m_cd144Stream.println("uStar: Surface Friction Velocity (cm/s)");
        m_cd144Stream.println("1/RaWDQI: Deposition velocity for Gas with Zo=f(WDQI) (centimeter/sec)");
        m_cd144Stream.println("1/RaOnshore: Deposition velocity for Gas with Zo=f(Onshore) (centimeter/sec)");
        m_cd144Stream.println("VdPM-Vg_WDQI: (1 / (Ra + Rd + Ra*Rd*Vg)) (cm/s)");
        m_cd144Stream.println("VdPM-Vg_Onshore: (1 / (Ra + Rd + Ra*Rd*Vg)) (cm/s)");
        m_cd144Stream.println("Vg: Gravitational Settling Velocity (cm/s)");
        m_cd144Stream.println("VdWDQI: Deposition Velocity with Zo=function(WDQI) (cm/s)");
        m_cd144Stream.println("VdOnshore: Deposition Velocity with Zo=function(Onshore) (cm/s)\n");
        m_cd144Stream.println("Diameter of Particulate Matter: "+m_opd.getDiaPM()+" microns
("+m_opd.getPMSizeRange()+")");
        m_cd144Stream.println("Density of Particulate Matter: "+m_opd.getRhoPM()+" microns");
        m_cd144Stream.println();
        m_cd144Stream.println("Surface Station Name: "+st.getName()+"-"+st.getCorresStation());
        m_cd144Stream.println("Surface Station Number: "+st.getNumber());
        m_cd144Stream.println();
        m_cd144Stream.println(HEADER_FIELDS);
        m_cd144Stream.println();
    }
    //-------------------------------------------------------------------------
    private void printValues()
    {
        String str = "";
        ArrayList lst = null;
        lst = m_opd.getDepositionRateValues();

        Iterator i = lst.iterator();
        while(i.hasNext())
            m_cd144Stream.print("\t"+str+i.next());
        lst = null;
    }
    //-------------------------------------------------------------------------
    private void convertAllVarToOutputFormat()
    {

        m_year = convertToOutputFormat(YEAR);
```

```
        m_month = convertToOutputFormat(MONTH);
        m_day = convertToOutputFormat(DAY);
        m_hour = convertToOutputFormat(HOUR);

        m_windDirectionInDegrees = convertToOutputFormat(WIND_DIRECTION_IN_DEGREES);
        m_windSpeed = convertToOutputFormat(WIND_SPEED);


        m_WDQI = convertToOutputFormat(WDQI);

        m_depositionVelocityFromBulkTransferCOef = convertToOutputFormat(DEPOSITION_VELOCITY); //21

        m_airPressure = convertToOutputFormat(A_PRESS);
        m_airRH = convertToOutputFormat(RH);
        m_airTemp = convertToOutputFormat(A_TEMP);
        m_waterTemp = convertToOutputFormat(W_TEMP);
        m_tempDiff = convertToOutputFormat(TEMP_DIFF);
        m_mixingRatio = convertToOutputFormat(MIXING_RATIO);

        if( tenOverL_Hack )
        {
            m_airVpTemp = convertToOutputFormat(30);
            m_waterVpTemp = convertToOutputFormat(31);

            VdPMminusVgWDQI = convertToOutputFormat(32);      //Z0_WDQI
            VdPMminusVgOnshore = convertToOutputFormat(33);   //Z0_Onshore
            Vg = convertToOutputFormat(35);                   //U10
        }
        else
        {
            m_airVpTemp = convertToOutputFormat(17);
            m_waterVpTemp = convertToOutputFormat(18);

            VdPMminusVgWDQI = convertToOutputFormat(25);
            VdPMminusVgOnshore = convertToOutputFormat(26);
            Vg = convertToOutputFormat(27);
        }

        m_airWaterVpTempDiff = convertToOutputFormat(19);
        m_bulkRichardsonNo = convertToOutputFormat(20);
        uStar = convertToOutputFormat(22);  //USTAR

        VdWDQI = convertToOutputFormat(28);
        VdOnshore = convertToOutputFormat(29);

    }


    //-----------------------------------------------------------------------
    private String convertToOutputFormat(int variable)
    {
        switch(variable){
```

```
            case 0:/***if year-------needs to be 5 digit no.***/
                return (""+cal.get(GregorianCalendar.YEAR));

            case 1: /***if month**----------needs to be 2 digit no.***/
                if(cal.get(GregorianCalendar.MONTH)+1 < 10) {
                       return "0" + (cal.get(GregorianCalendar.MONTH)+1);
                }
                else{
                        return "" + (cal.get(GregorianCalendar.MONTH)+1);
                }

            case 2: /****if day ---------needs to be 2 digit no.******/
                if(cal.get(GregorianCalendar.DAY_OF_MONTH)<10) {
                    return "0" + cal.get(GregorianCalendar.DAY_OF_MONTH);
                }
                else{
                    return "" + cal.get(GregorianCalendar.DAY_OF_MONTH);
                }

            case 3: /**if hour -------needs to be 2 digit no.***/

                int hrInt = cal.get(GregorianCalendar.HOUR_OF_DAY);
                String hourString = ""+hrInt;

                if(hourString.length() == 1){
                    return "0"+ hourString;
                }
                else if(hourString.length() == 2){
                    return hourString;
                }

                return hourString;

            case 5: /***if average wind direction-------in degrees*******/

                if(avgWind == null || (! avgWind.isDataSet()) )
                    return "";
                else{

                    return ""+(new BigDecimal(""+avgWind.getWindDirectionInDegrees())).setScale(3,
BigDecimal.ROUND_HALF_EVEN);
                }

            case 6:/***if average wind speed----- in knots--*******/

                if(avgWind == null || (! avgWind.isDataSet()))
                    return "";
                else {
                    return "" + (new BigDecimal(""+avgWind.getWindSpeedInMetersPerSecond())).setScale(3,
BigDecimal.ROUND_HALF_EVEN);
                }

            case 9: /**if WDQI*/
                if(avgWind == null || (! avgWind.isDataSet()))
```

```
                return "";
            else {
                return "" + avgWind.getWDQI();
            }
        case 10: /**if Air pressure------------in miliBars-------------------*/
            if(avgAir == null || (! avgAir.isDataSet()))
                return "";
            else {
                return "" + (new BigDecimal(""+avgAir.getAvgAirPressure())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
            }
        case 11: /**if Air RH*/
            if(avgAir == null || (! avgAir.isDataSet()))
                return "";
            else {
                return "" + (new BigDecimal(""+avgAir.getAvgRH())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
            }

        case 12: /**if Air Temp-----------in degree celcius----------------*/
            if(avgAir == null || (! avgAir.isDataSet()))
                return "";
            else {
                return "" + (new BigDecimal(""+avgAir.getAvgAirTempDegreeC())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
            }

        case 13: /**if Water Temp-----------in degree celcius----------------*/
            if(avgWater == null || (!avgWater.isDataSet()))
                return "";
            else {
                return "" + (new BigDecimal(""+avgWater.getAvgTempDegreeC())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
            }

        case 14: /**if Temp Difference then calculate Air Temp - Water Temp---in degree celcius------*/
            double airTemp = 0;
            if(avgAir == null || (! avgAir.isDataSet()))
                return "";
            else {
                airTemp = avgAir.getAvgAirTempDegreeC();
            }

            double waterTemp = 0;
            if(avgWater == null || (! avgWater.isDataSet()))
                return ""+(new BigDecimal(""+airTemp)).setScale(3, BigDecimal.ROUND_HALF_EVEN);
            else {
                waterTemp = avgWater.getAvgTempDegreeC();
            }
            return ""+ (new BigDecimal(""+ (airTemp - waterTemp))).setScale(3, BigDecimal.ROUND_HALF_EVEN);


        case 15:
            if(avgAir == null || (! avgAir.isDataSet()))
                return "";
            else {
                return "" + (new BigDecimal(""+avgAir.getMixingRatio())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
```

```
                }

            case 17: /*****if Air VpTemp. ***-----------in degree kelvin----------------*****/
                if(! m_opd.isDataSet())
                    return "";
                else {
                    return "" + (new BigDecimal(""+avgAir.getVirtualPotentialTempDegreeK())).setScale(3,
BigDecimal.ROUND_HALF_EVEN);
                }
            case 18: /*****if Water VpTemp. **-----------in degree kelvin----------------******/
                if(! m_opd.isDataSet())
                    return "";
                else {
                    return "" + (new BigDecimal(""+m_opd.getVirtualPotentialTempAtSurfaceDegreeK())).setScale(3,
BigDecimal.ROUND_HALF_EVEN);  /* ERROR */
                }
            case 19: /*****if Air Water VpTemp. Difference **-----------in degree kelvin----------------******/
                if(! m_opd.isDataSet())
                    return "";
                else {
                    return "" + (new BigDecimal(""+m_opd.getVirtualPotentialAirWaterTempDiff())).setScale(3,
BigDecimal.ROUND_HALF_EVEN);
                }
            case 20: /*****if Bulk Richardson number ********/
                if(! m_opd.isDataSet())
                    return "";
                else {
                    return "" + (new BigDecimal(""+m_opd.getBulkRichardsonNo())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
                }
            case 21: /***if Deposition Velocity from Bulk Transfer Coeff***** the other type is case 16********/
                if(avgWind == null || (! avgWind.isDataSet()) )
                    return "";
                else {
//                  return "" + (new BigDecimal(""+avgWind.getDepositionVelocityFromBulkTransferCOeff())).setScale(3,
BigDecimal.ROUND_HALF_EVEN);    // Uses U10 from avgWind
//                  System.out.println("!!!!:"+m_opd.getU10()+" !!!:"+avgWind.getDepositionVelocityFromBulkTransferCOeff(
m_opd.getU10() ));  // KILL THIS LATER
                    try {
                        return "" + (new BigDecimal(""+avgWind.getDepositionVelocityFromBulkTransferCOeff( m_opd.getU10()
))).setScale(3, BigDecimal.ROUND_HALF_EVEN);  // USE UStar from the output data creator instead of U10!
                    } catch( Exception e ) {
                        return "";
                    }
                }

            case 22: /***if Deposition Velocity from Station Height***** the other type is case 16********/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {  // get UStar!!
                    return "" + (new BigDecimal(""+ m_opd.getuStar())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
                }
            case 24: /***if Deposition Velocity from Station Height***** the other type is case 16********/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
```

```
                    return "" + (new BigDecimal(""+ m_opd.getVg())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
                }
            case 25: /***if VdPM - Vg for WDQI********/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + (new BigDecimal(""+ m_opd.getVdPMminusVgForWDQI())).setScale(7,
BigDecimal.ROUND_HALF_EVEN);
                }
            case 26: /***if VdPM - Vg for Onshore********/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + (new BigDecimal(""+ m_opd.getVdPMminusVgForOnshore())).setScale(7,
BigDecimal.ROUND_HALF_EVEN);
                }
            case 27: /***if  Vg********/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + (new BigDecimal(""+ m_opd.getVg())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
                }
            case 28: /***if VdWDQI********/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + (new BigDecimal(""+ m_opd.getVdForWDQI())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
                }
            case 29: /***if VdOnshore********/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + (new BigDecimal(""+ m_opd.getVdForOnshore())).setScale(3, BigDecimal.ROUND_HALF_EVEN);
                }
            case 30: /** if L for WDQI  **/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + (10.0/m_opd.getL_ForWDQI());
                }
            case 31: /** if L for Onshore  **/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + (10.0/m_opd.getL_ForOnshore());
                }
            case 32: /** if Z0 for WDQI  **/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + m_opd.getZ0_ForWDQI();
                }
            case 33: /** if Z0 for Onshore  **/
                if(m_opd == null || (! m_opd.isDataSet()))
```

```
                    return "";
                else {
                    return "" + m_opd.getZ0_ForOnshore();
                }
            case 35: /** if U10  **/
                if(m_opd == null || (! m_opd.isDataSet()))
                    return "";
                else {
                    return "" + m_opd.getU10();    // Uses U10 from avgWind
                }

        }
        return "";
    }
}
```

AvgAirTempReader.java –

Reads in a text file containing hourly all-lake air temperature information, providing this data when the input file is missing data

```
package Batch2;

import java.util.*;
import java.util.List;
import java.io.*;

/**
 * AverageAirTempReader - Class for managing
 *  information contained in a flat file
 *  related to the average air temperature.
 *  Data is stored/read according to a specific
 *  format, and becomes addressable by
 *  year, month, day, and hour.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 */
public class AvgAirTempReader {

    private String         myFileName;
    private double[][][][] myData;
    private boolean        validData;

    public static final double GARBAGE = -6999;

    public AvgAirTempReader(String s) {
        myFileName = s;
        validData = false;
        myData = new double[2][12][31][24];

        /*  initialize the data to "AvgTempReader.GARBAGE"  */
        for( int i1 = 0; i1 < 2; i1++ )
            for( int i2 = 0; i2 < 12; i2++ )
                for( int i3 = 0; i3 < 31; i3++ )
                    for( int i4 = 0; i4 < 24; i4++ )
                        myData[i1][i2][i3][i4] = GARBAGE;
        System.out.print("Loading Air_Temp Data from: "+s+" ...");
        initializeData();
        System.out.println("Done! ");
    }
        /* check the Valid Data flag */
    public boolean hasValidData() {
        return validData;
    }

        /* Initialize the data in the object from the file */
    private void initializeData() {
        try {
            BufferedReader  inFile = new BufferedReader( new FileReader( myFileName ) );
            StringTokenizer t;
            boolean startParse = false;
```

```
            for( int i = 0 ; ((i < 30) && !startParse)  ; i++ )
            {
                t = new StringTokenizer( inFile.readLine(), " \t" );
                while( t.hasMoreTokens() )
                   if( t.nextToken().equals( "Tmax" ) )
                       if( t.nextToken().equals( "Tmax(site)" ) )
                           startParse = true;
            }
            if( !startParse )
            {
                System.out.println("Error parsing Air_Temp file: No start of data.");
                return;
            }
            String myLine;
            while( (myLine = inFile.readLine() ) != null ) {
                t = new StringTokenizer( myLine , "\t " );
                int yy,mm,dd,hh;
                double Tavg;
                yy = Integer.parseInt( t.nextToken() );
                mm = Integer.parseInt( t.nextToken() );
                dd = Integer.parseInt( t.nextToken() );
                hh = Integer.parseInt( t.nextToken() );

                if( ((myLine=t.nextToken()).equals("-1.#IND")) )
                    Tavg = GARBAGE;
                else
                    Tavg = Double.parseDouble( myLine );

                myData[yy-2002][mm-1][dd-1][hh] = Tavg;

                    /* go to the next line */
            }
           validData = true;  // File has been parsed correctly!
            return;            //  done!!
        } catch( Exception e ) {
            System.out.println( "Exception in AvgAirTempReader: "+e);
        }
    }

    public boolean isValidData( int YY,int MM,int DD, int HH ) {
        return( myData[YY-2002][MM-1][DD-1][HH] != GARBAGE );
    }

    public double getData( int YY,int MM,int DD, int HH ) {
        try {
            return  myData[YY-2002][MM-1][DD-1][HH];
        } catch (Exception e ) {
            return GARBAGE;
        }
    }
}
```

AvgWaterTempReader.java –

Reads in a text file containing hourly all-lake water temperature information, providing this data when the input file is missing data

```
package Batch2;

import java.util.*;
import java.util.List;
import java.io.*;

/**
 * AverageWaterTempReader - Class for managing
 *  information contained in a flat file
 *  related to the average water temperature.
 *  Data is stored/read according to a specific
 *  format, and becomes addressable by
 *  year, month, day, and hour.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 */
public class AvgWaterTempReader {

    private String         myFileName;
    private double[][][][] myData;
    private boolean        validData;

    public static final double GARBAGE = -6999;

    public AvgWaterTempReader(String s) {
        myFileName = s;
        validData = false;
        myData = new double[2][12][31][24];

        /*  initialize the data to "AvgTempReader.GARBAGE"  */
        for( int i1 = 0; i1 < 2; i1++ )
            for( int i2 = 0; i2 < 12; i2++ )
                for( int i3 = 0; i3 < 31; i3++ )
                    for( int i4 = 0; i4 < 24; i4++ )
                        myData[i1][i2][i3][i4] = GARBAGE;
        System.out.print("Loading H2O Data from: "+s+" ...");
        initializeData();
        System.out.println("Done! ");
    }
        /* check the Valid Data flag */
    public boolean hasValidData() {
        return validData;
    }

        /* Initialize the data in the object from the file */
    private void initializeData() {
        try {
            BufferedReader  inFile = new BufferedReader( new FileReader( myFileName ) );
            StringTokenizer t;
            boolean startParse = false;
```

```
        for( int i = 0 ; ((i < 30) && !startParse)  ; i++ )
        {
            t = new StringTokenizer( inFile.readLine(), " \t" );
            while( t.hasMoreTokens() )
            {
                if( t.nextToken().equals( "Tmax" ) )
                    if( t.nextToken().equals( "Tmax(site)" ) )
                        startParse = true;
            }

        }
        if( !startParse )
        {
            System.out.println("Error parsing H2O file: No start of data.");
            return;
        }

        String myLine;
        while( (myLine = inFile.readLine() ) != null ) {
            t = new StringTokenizer( myLine , "\t " );
            int yy,mm,dd,hh;
            double Tv;
            yy = Integer.parseInt( t.nextToken() );
            mm = Integer.parseInt( t.nextToken() );
            dd = Integer.parseInt( t.nextToken() );
            hh = Integer.parseInt( t.nextToken() );

             /* ignore next 8 fields */
            t.nextToken();
            t.nextToken();
            t.nextToken();
            t.nextToken();
            t.nextToken();
            t.nextToken();
            t.nextToken();
            t.nextToken();

            if( ((myLine=t.nextToken()).equals("#VALUE!")) )
                Tv = GARBAGE;
            else
                Tv = Double.parseDouble( myLine );

            myData[yy-2002][mm-1][dd-1][hh] = Tv;
        }

        validData = true;  // File has been parsed correctly!
        return;            //  done!!


    } catch( Exception e ) {
        System.out.println( "Exception in AvgTempReader: "+e);
    }
```

```
    }

    public boolean isValidData( int YY,int MM,int DD, int HH ) {
        return( myData[YY-2002][MM-1][DD-1][HH] != GARBAGE );
    }

    public double getData( int YY,int MM,int DD, int HH ) {
        try {
            return  myData[YY-2002][MM-1][DD-1][HH];
        } catch (Exception e ) {
            return GARBAGE;
        }
    }




}
```

PropertiesViewer.java –

Contains helper methods for accessing a properties file and reading/parsing types of information

```
package Batch2;

import java.lang.*;
import java.util.*;

/**
 * PropertiesViewer - Utilitiy class for retrieving
 *  data from either a Properties file.  May parse
 *  a value from a key as different primitives or as
 *  a string.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 */
public class PropertiesViewer {

    private String                myFile;
    private PropertyResourceBundle myBundle;


    public PropertiesViewer( String readFile )
    {
        try {
            myFile = readFile;
            myBundle = (PropertyResourceBundle)(ResourceBundle.getBundle( readFile ));
        } catch( Exception e0 ) {
            myFile = S_ERR;
            myBundle = null;
        }
    }
    private PropertyResourceBundle getBundle() {
        return myBundle;
    }

    public boolean validBundle() {
        return ( myBundle != null );
    }

    public int getInt( String param ) {
        try {
            return( Integer.parseInt( myBundle.getString(param).trim() ) );
        } catch ( Exception e ) {
            return I_ERR;
        }
    }
    public double getDouble( String param ) {
        try {
            return( Double.parseDouble(myBundle.getString(param).trim()) );
        } catch ( Exception e ) {
            return D_ERR;
        }
```

```
    }
    public String getString( String param ) {
        try {
            return( myBundle.getString(param));
        } catch ( Exception e ) {
            return S_ERR;
        }
    }
    public boolean getBoolean( String param ) {
        try {
            return( Boolean.valueOf( myBundle.getString(param).trim() ).booleanValue() );
        } catch ( Exception e ) {
            return false;
        }
    }

    public static int getInt( String filename, String param ) {
        try {
            return( Integer.parseInt((new PropertiesViewer(filename)).getBundle().getString(param).trim()) );
        } catch ( Exception e ) {
            return I_ERR;
        }
    }
    public static double getDouble( String filename, String param ) {
        try {
            return( Double.parseDouble((new PropertiesViewer(filename)).getBundle().getString(param)) );
        } catch ( Exception e ) {
            return D_ERR;
        }
    }
    public static String getString( String filename, String param ) {
        try {
            return( ((new PropertiesViewer(filename)).getBundle().getString(param))   );
        } catch ( Exception e ) {
            System.out.println( "Exception: "+e);
            return S_ERR;
        }
    }
    public static boolean getBoolean( String filename , String param ) {
        try {
            return( Boolean.valueOf( (new PropertiesViewer(filename)).getBundle().getString(param).trim() ).booleanValue()
);
        } catch ( Exception e ) {
            return false;
        }
    }


    public static final int    I_ERR = -6669;
    public static final String S_ERR = "#NO_DATA";
    public static final double D_ERR = -3.31981571861;

}
```

OutputSummarizer.java –

Calculates seasonal averages for various columns of a .dpR file, adding the information to the header cells

```
/*
 * OutputFileSummarizer.java
 *
 * Created on December 30, 2004, 12:02 PM
 */
package Batch2;

import java.io.*;
import java.lang.*;
import java.util.*;

/**
 * OutputFileSummarizer - Class for processing
 *  output files after they are generated
 *  by the Main module.  Generates Summary information
 *  by season, or hourly by season.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @see Batch2.SummarySummarizer
 * @see Batch2.PropertiesViewer
 */
public abstract class OutputFileSummarizer {

    public static final String CONFIG_FILE = Main.CONFIG_FILE;

    /** Calculates the seasonal averages for various parameters
     *  in the file.  Will recalculate the Annual Average based
     *  off of the available seasons if <code>RecalculateAnnualOnSeasonal</code>
     *  is set in the <code>CONFIG_FILE</code>
     *  @param filename The file to perform the summary on.
     */
    public static void doSummary( String filename )
    {
        File inputFile  = new File(filename);
        File tempFile   = new File(filename+".tmp");
        System.out.print("Calculating Summary Data...");
        createTempFromInput( inputFile, tempFile );
        inputFile.delete();
        tempFile.renameTo(inputFile);
        if( PropertiesViewer.getBoolean(CONFIG_FILE,"RecalculateAnnualOnSeasonal"))
        {
            System.out.print("_");
            SummarySummarizer.recalculateAnnualFromSeasons(filename);
        }


        System.out.println("complete!");
    }
```

```
    /** Calculates the hourly seasonal averages for various parameters
     *  in the file.
     *  @param filename The file to perform the summary on.
     */
    public static void doSummary_Hourly( String filename )
    {
        File inputFile  = new File(filename);
        File tempFile   = new File(filename+".hourly");
        createTempFromInput_Hourly( inputFile, tempFile );
    }



        /***********************
         **  SEASONAL METHOD  **
         **********************/
    private static void createTempFromInput( File inputFile, File outputFile )
    /* This does most of the work!!  */
    {
        BufferedReader fin  = null;
        PrintStream    fout = null;
          /** Open the Streams **/
        try {
            fin  = new BufferedReader( new FileReader( inputFile ));
            fout = new PrintStream( new FileOutputStream(outputFile) );
        } catch( Exception e ) {
            System.out.println("Error with File Streams: "+e);
            System.exit( 0 );
        }


          /** Perform the Action **/
        String s = null;
        try {
            boolean found = false;
            int i,j;

               /** Read past the header cells **/
            if( !readInclusiveUntil(fin, "YYYY\tMM\tDD\tHH\t") )
                throw new Exception("Start of Data not Found!");

            /***************************
             *
             * Gather the Summary Info!!!!!!!
             *
             ***************************/
            double[][] myArray = new double[5][26];
            double temp;
            int[] counter = new int[5];
            for( j = 0; j < 5 ; j++)
            {
                counter[j] = 0;
                for( i = 0; i < 26 ; i++)
```

```
                myArray[j][i]=0.0;
            }
            StringTokenizer t;
            while( (s = fin.readLine()) != null )
            {
                if( (t = new StringTokenizer(s,"\t")).countTokens() >= 32 )
                {
                      /** increment the annual index **/
                    counter[0]++;
                      /** get the season, increment the index **/
                    skip(t,1); // skip YY
                    j = seasonIndex( t.nextToken() );
                    counter[j]++;


                      /** Add to Wind Speed **/
                    skip(t,2); // skip DD,HH
                    myArray[0][0]+=(temp=Double.parseDouble( t.nextToken()));
                    myArray[j][0]+=temp;
                    skip(t,2); // skip WD,WDQI

                      /** Add to other values **/
                    for( i = 0; i < 25; i++)
                    {
                        myArray[0][i+1]+=(temp=Double.parseDouble(t.nextToken()));
                        myArray[j][i+1]+=temp;
                    }
                }
            }

            for( j = 0; j < 5; j++)
                for( i = 0; i < myArray[j].length; i++)
                {
                    if( counter[j] != 0 )
                        myArray[j][i] /= counter[j];
                    else
                        myArray[j][i] = -6999.0;
                }


                /** Reset the input stream!! **/
            fin.close();
            try { fin  = new BufferedReader( new FileReader( inputFile ));
            } catch( Exception e ) {
                System.out.println("Error with File Streams: "+e);
                System.exit( 0 );
            }

            /** Reprint Headers **/
            reprint( fin, fout, 40 );
            skip( fin, 1 );

            /****************************
```

```
            *
            * Print the Summary Info!!!!!!!
            *
            ***************************/
           for( i = 0; i < 5; i++)
           {
               fout.print( "\t\t\t\t"+str(myArray[i][0])+"\t\t"+seasonName(i)+"\t" );
               for( j = 1; j < myArray[i].length; j++)
                   fout.print( str(myArray[i][j])+"\t" );
               fout.println( seasonName(i)+"\t"+counter[i]+"\t" );
           }

           readExclusiveUntil(fin,"YYYY\tMM\tDD\tHH\t");
           reprint( fin, fout );  /** reprint the headers / data **/

       } catch( Exception e ) { System.out.println(""+e);  }
//       fout.println( "File Parsed by OutputFileSummarizer" );

       try {  fin.close();  } catch( Exception e2 ) { System.out.println(""+e2);  }
       try {  fout.close(); } catch( Exception e3 ) { System.out.println(""+e3);  }
   }





       /********************
        **  HOURLY METHOD  **
        ********************/

   private static void createTempFromInput_Hourly( File inputFile, File outputFile )
   {
       BufferedReader fin  = null;
       PrintStream    fout = null;
         /** Open the Streams **/
       try {
           fin  = new BufferedReader( new FileReader( inputFile ));
           fout = new PrintStream( new FileOutputStream(outputFile) );
       } catch( Exception e ) {
           System.out.println("Error with File Streams: "+e);
           System.exit( 0 );
       }


         /** Perform the Action **/
       String s = null;
       try {
           boolean found = false;
           int i,j,k;

              /** Read past the header cells **/
           if( !readInclusiveUntil(fin, "YYYY\tMM\tDD\tHH\t") )
               throw new Exception("Start of Data not Found!");
```

```
/****************************
 * Gather the Summary Info!!!!!!!
 ****************************/
double[][][] myArray = new double[5][25][26];
double temp;
int[][] counter = new int[5][25];
for( j = 0; j < 5 ; j++)
{
    for( k = 0; k < 25 ; k++)
    {
        counter[j][k] = 0;
        for( i = 0; i < 26 ; i++)
            myArray[j][k][i]=0.0;
    }
}
StringTokenizer t;
while( (s = fin.readLine()) != null )
{
    if( (t = new StringTokenizer(s,"\t")).countTokens() == 32 )
    {
          /** get the season and time **/
        skip(t,1); // skip YY
        j = seasonIndex( t.nextToken() );
        skip(t,1); // skip DD,HH
        k = Integer.parseInt(t.nextToken())+1;

        counter[0][0]++;
        counter[j][0]++;
        counter[0][k]++;
        counter[j][k]++;


          /** Add to Wind Speed **/
        myArray[0][0][0]+=(temp=Double.parseDouble( t.nextToken()));
        myArray[j][0][0]+=temp;
        myArray[0][k][0]+=temp;
        myArray[j][k][0]+=temp;
        skip(t,2); // skip WD,WDQI

          /** Add to other values **/
        for( i = 0; i < 25; i++)
        {
            myArray[0][0][i+1]+=(temp=Double.parseDouble(t.nextToken()));
            myArray[j][0][i+1]+=temp;
            myArray[0][k][i+1]+=temp;
            myArray[j][k][i+1]+=temp;
        }
    }
}

for( j = 0; j < 5; j++)
    for( k = 0; k < 25; k++)
```

```
                    for( i = 0; i < 26; i++)
                    {
                        if( counter[j][k] != 0 )
                            myArray[j][k][i] /= counter[j][k];
                        else
                            myArray[j][k][i] = -6999.0;
                    }


                /** Reset the input stream!! **/
            fin.close();
            try { fin  = new BufferedReader( new FileReader( inputFile ));
            } catch( Exception e ) {
                System.out.println("Error with File Streams: "+e);
                System.exit( 0 );
            }

            /** Reprint Headers **/
            reprint( fin, fout, 40 );
//            skip( fin, 1 );

            // Print the Summary Info //
            for( i = 0; i < 5; i++)
            {
                for( k = 0; k < 25 ; k++ )
                {
                    fout.print( "\t\t\t\t"+str(myArray[i][k][0])+"\t"+timeName(k)+"\t"+seasonName(i)+"\t" );
                    for( j = 1; j < 26; j++)
                        fout.print( str(myArray[i][k][j])+"\t" );
                        fout.println( "\t"+timeName(k)+"\t"+seasonName(i)+"\t"+counter[i][k]+"\t" );
                }
            }

            readExclusiveUntil(fin,"YYYY\tMM\tDD\tHH\t");
//          reprint( fin, fout );  /** reprint the headers / data **/
            reprint( fin, fout, 1 );  /** reprint the headers (but no data) **/

        } catch( Exception e ) { System.out.println(""+e);  }
//        fout.println( "File Parsed by OutputFileSummarizer" );

        try {  fin.close();  } catch( Exception e2 ) { System.out.println(""+e2);  }
        try {  fout.close(); } catch( Exception e3 ) { System.out.println(""+e3);  }
    }
```

```
private static void reprint( BufferedReader in, PrintStream out, int numLines )
{
    String s;
    try {
        for( int i = 0; i < numLines && (s = in.readLine()) != null ; i++ )
            out.println( s );
    } catch( Exception E ) { System.out.println(""+E); }
}

private static void reprint( StringTokenizer in, PrintStream out, int numLines )
{
    String s;
    try {
        for( int i = 0; i < numLines && (s = in.nextToken()) != null ; i++ )
            out.print( s );
    } catch( Exception E ) {  }
}

private static void reprint( BufferedReader in, PrintStream out )
{
    String s;
    try {
        while ((s = in.readLine()) != null)
            out.println( s );
    } catch( Exception E ) { System.out.println(""+E);  }

}

private static void skip( BufferedReader in , int numLines )
{
    try {
        for( int i = 0; i < numLines ; i++ )
            in.readLine();
    } catch( Exception E ) { System.out.println(""+E);  }
}

private static void skip( StringTokenizer t, int numTokens )
{
    for( int i = 0; i < numTokens; i++ )
        t.nextToken();
}



private static String str( double d )
{
    if( d == -6999.0 )
        return "";
    else
        return ""+d;
}
```

```
private static boolean readInclusiveUntil( BufferedReader br, String s )
    throws IOException
{
    String ss="";
    while( ss != null )
    {
        if( (ss=br.readLine()).indexOf(s) != -1 )
            return true;
    }
    return false;
}
private static boolean readExclusiveUntil( BufferedReader br, String s )
    throws IOException
{
    String ss="";
    while( ss != null )
    {
        br.mark( 750 );
        if( (ss=br.readLine()).indexOf(s) != -1 )
        {
            br.reset();
            return true;
        }
    }
    return false;
}

private static int seasonIndex( String s )
{
    try {
        int i = Integer.parseInt( s );
        switch( i )
        {
            case 12: case 1: case 2:
                return 4;
            case 3: case 4: case 5:
                return 1;
            case 6: case 7: case 8:
                return 2;
            case 9: case 10: case 11:
                return 3;
            default:
                return 0;
        }
    } catch ( Exception e ) {
        return 0;
    }
}
private static String seasonName( int i )
```

```
    {
        switch( i )
        {
            case 0:
                return "Annual";
            case 1:
                return "Spring";
            case 2:
                return "Summer";
            case 3:
                return "Fall";
            case 4:
                return "Winter";
            default:
                return "";
        }
    }

    private static String timeName( int k ) {
        switch( k )
        {
            case 0:
                return "All-Day";
            default:
                return ""+(k-1);
        }
    }



}
```

SummarySummarizer.java –

Calculates parameters based on a group of .dpr files; creates a combined workbook.

```
/*
 * SummarySummarizer.java
 *
 * Created on December 31, 2004, 2:43 PM
 */

package Batch2;

import java.util.*;
import java.io.*;
import java.lang.*;
import org.apache.poi.hssf.usermodel.*;



/**
 *
 * @author  pruibal
 */
public class SummarySummarizer {
    /** Creates a new instance of SummarySummarizer */
    public SummarySummarizer() {      }


    public void deletePages( String[] pages )
    {


    }



   private static int seasonIndex( String season )
   {
      switch( season.charAt( 1 ) )
      {
         case 'n':      // Annual
            return 0;
         case 'p':      // Spring
            return 1;
         case 'u':      // Summer
            return 2;
         case 'a':      // Fall
            return 3;
         case 'i':      // Winter
            return 4;
      }
      return -1;

   }
```

```
    private static void skip( StringTokenizer t, int numTokens )
    {
        String s;
        for( int i = 0; i < numTokens; i++ )
            s = t.nextToken();
    }
    private static void skip( BufferedReader br, int numLines )
    {
        try {
            for( int i = 0; i < numLines; i++ )
                br.readLine();
        } catch( IOException e ) { System.out.println("Exception: "+e); }
    }
    private static void copyLines( BufferedReader br, PrintStream pw, int numLines )
    {
        try {
            String S;
            for( int i = 0; i < numLines; i++ )
                if( (S = br.readLine())!= null )
                    pw.println(S);
        } catch( IOException e ) { System.out.println("Exception: "+e); }
    }
    private static void reprint( BufferedReader in, PrintStream out )
    {
        String s;
        try {
            while ((s = in.readLine()) != null)
                out.println( s );
        } catch( Exception E ) { System.out.println(""+E);  }

    }

    private static void skipUntil( BufferedReader br, String myStr )
    {
        try {
            String s;
            br.mark( 2000 );
            while( (s=br.readLine())!= null ) {
                if( s.indexOf(myStr) != -1 )
                    break;
                br.mark( 2000 );
            }
            br.reset();
        } catch (Exception e) { System.out.println("Exception: "+e); }
    }



    private static String replace( String in, String change, String with )
    {
   int i;
        while( (i=in.indexOf(change)) != -1 )
        {
```

```
//            System.out.print(".");
            in = "" + in.substring(0,i) + with + in.substring(i+change.length());
        }
        return in;
    }



  private static boolean readInclusiveUntil( BufferedReader br, String s )
        throws IOException
    {
        String ss="";
        while( ss != null )
        {
            if( (ss=br.readLine()).indexOf(s) != -1 )
                return true;
        }
        return false;
    }


      /************************
       **  SEASONAL METHODS  **
       ************************/

   public static void combineWorksheets( String[] pages, String[] titles, String workbookname )
    {
        try {
            /** create new workbook object **/
            HSSFWorkbook myWorkbook = new HSSFWorkbook();
            HSSFSheet currentSheet;

            HSSFFont  F_bold  = myWorkbook.createFont();
            F_bold.setBoldweight(HSSFFont.BOLDWEIGHT_BOLD );
            HSSFCellStyle S_bold  = myWorkbook.createCellStyle();
            S_bold.setFont(F_bold);


            for( int i = 0; i < pages.length; i++ )
            {
                /** create the BufferedReader for each page **/
                BufferedReader br=new BufferedReader(new FileReader(pages[i]));
                int row=0,col=0;
                String s;

                /** parse the report pages into workbook sheets **/
                currentSheet = myWorkbook.createSheet(titles[i]);

                while( (s=br.readLine()) != null )
                {
                    s = replace(s,"\t"," @");
                    StringTokenizer t = new StringTokenizer( s, "@");
                    HSSFRow myRow = currentSheet.createRow((short)row++);
```

```
                    col=0;
                    while( t.hasMoreTokens() )
                    {
                        HSSFCell cell = myRow.createCell((short)col++);
                        if( row-1 == 0 || (row-2)%7 == 0 || ((row-1 > 21) && (col==3))  )
                            cell.setCellStyle(S_bold);

                        String ss = t.nextToken();
                        try {
                            cell.setCellValue( Double.parseDouble(ss) );
                        } catch( Exception e ) {
                            if( ss.charAt(0) == '=' )
                                cell.setCellFormula( ss );
                            else
                                cell.setCellValue(ss);
                        }
//                        cell.setCellValue(t.nextToken());

                    }
                }

                System.out.print(".");
            }
            /** Write the workbook object out to (workbookname) file **/
            FileOutputStream fileOut = new FileOutputStream(workbookname);
            myWorkbook.write(fileOut);
            fileOut.close();
        } catch( Exception e ) {
            System.out.println("Exception: "+e); e.printStackTrace();
        }
    }




    public static void recalculateAnnualFromSeasons( String sourceFile )
    {
        try {
            BufferedReader br1 = new BufferedReader( new FileReader( sourceFile ) );
            String [][] sourceData;
            double []   totals;
            int    []   counts;
            String curLine;
            StringTokenizer t1,t2;


            /**   Initialize Data Structures    **/
            int i,j;
            sourceData    = new String[5][40];
            totals = new double[40];
            counts = new int[40];
            for( i = 0 ; i < 5 ; i++ )
            {
```

```
                counts[i] = 0;
                totals[i] = 0.0;
                for( j = 0 ; j < 40 ; j++ )
                    sourceData[i][j] = "#";
            }

            /**   Read source Data from the file **/
            skipUntil( br1, "Annual" );  //exclusive
            int numTokens = 0;
            for( i = 0 ; i < 5 ; i++ )
            {
                t1 = new StringTokenizer( br1.readLine(), "\t" );
                j = 0;
                while( t1.hasMoreTokens() )
                    sourceData[i][j++] = t1.nextToken();
                if( i == 0 )   numTokens = j;  // save the number of fields //
            }


            /**   Get Counts/Totals from the sourceData   **/
            for( i = 1 ; i < 5 ; i++ )
            {
                if( sourceData[i][0].equals("#") || sourceData[i][0].length() < 1 )
                    continue;
                else
                    for( j = 0 ; j < numTokens ; j++ )
                    {
//                      System.out.println(sourceData[i][j]);
                        try {
                            totals[j] += Double.parseDouble(sourceData[i][j]);
                            counts[j]++;
                        } catch( NumberFormatException e2 ) { }
                    }
            }

            /**   calculate final sourceData from saved totals/counts   **/
            for( j = 0; j < numTokens; j++ )
            {
                try {
                    if( counts[j] == 0 )
                        continue;
                    Double.parseDouble( sourceData[0][j] );
                    sourceData[0][j] = "" + totals[j]/counts[j];
                } catch( NumberFormatException e2 ) { }
            }

            br1.close();


            /**   Write sourceData to sourceFile.tmp   **/
            br1 = new BufferedReader( new FileReader( sourceFile ) );
            PrintStream ps = new PrintStream( new FileOutputStream( sourceFile+".tmp" ) );
            copyLines( br1,ps,40 );
```

```
            for( i = 0; i < 5; i++ )
            {
                ps.print("\t\t\t\t"+sourceData[i][0]+"\t\t");
                for( j = 1; j < numTokens; j++ )
                {
                    if(sourceData[i][j].equals("#") )
                        sourceData[i][j] = "";
                    ps.print(sourceData[i][j]+"\t");
                }

                ps.print("\n");
            }
            skipUntil( br1,"YYYY" );
            reprint( br1,ps );
            br1.close();
            ps.close();


            /**   Replace old file with .tmp file  **/
            File inputFile  = new File(sourceFile);
            File tempFile   = new File(sourceFile+".tmp");
            inputFile.delete();
            tempFile.renameTo(inputFile);


        } catch ( Exception e ) {
            e.printStackTrace();
        }
    }


    public static void calculateSurrogateData( String sourceFile, String surrogateFile, String badSeason, String
goodSeasons )
    {
        try {
            BufferedReader br1 = new BufferedReader( new FileReader( sourceFile ) );
            BufferedReader br2 = new BufferedReader( new FileReader( surrogateFile ) );
            String [][] sourceData, surrogateData;
            String curLine;
            StringTokenizer t1,t2;


            /**   Initialize Data Structures    **/
            int i,j;
            sourceData    = new String[5][40];
            surrogateData = new String[5][40];

            for( i = 0 ; i < 5 ; i++ )
            {
                for( j = 0 ; j < 40 ; j++ )
                {
                    sourceData[i][j] = "";
                    surrogateData[i][j] = "";
                }
```

```
            }


            /**   Read Data from Files    **/
            skipUntil( br1, "Annual" );
            skipUntil( br2, "Annual" );
            int numTokens = 0;
            for( i = 0 ; i < 5 ; i++ )
            {
                t1 = new StringTokenizer( br1.readLine(), "\t" );
                j = 0;
                while( t1.hasMoreTokens() )
                    sourceData[i][j++] = t1.nextToken();

                t2 = new StringTokenizer( br2.readLine(), "\t" );
                j = 0;
                while( t2.hasMoreTokens() )
                    surrogateData[i][j++] = t2.nextToken();

                if( i == 0 )   numTokens = j;  // save the number of fields //
            }

            br1.close();
            br2.close();



            /**   calculate final sourceData from sourceData,surrogateData,
            badSeason,goodSeasons   **/

            int numGoodSeasons = (t2=new StringTokenizer( goodSeasons, ", " )).countTokens();

            for( i = 0 ; i < numTokens ; i++ )  // for each field in the dpr file... //
            {
                try {
                    double  behaviorSurrogate=0.0,
                    behaviorSource=0.0;

                    // get averaged source (bad) data //

                    t2 = new StringTokenizer( goodSeasons, ", " );
                    while( t2.hasMoreTokens() ) // for each "good season"...
                        behaviorSource += Double.parseDouble(sourceData[seasonIndex(t2.nextToken())][i]);
                    behaviorSource /= (double)numGoodSeasons;

                    // get averaged surrogate (good) data //
                    t2 = new StringTokenizer( goodSeasons, ", " );
                    while( t2.hasMoreTokens() ) // for each "good season"...
                        behaviorSurrogate += Double.parseDouble(surrogateData[seasonIndex(t2.nextToken())][i]);
                    behaviorSurrogate /= (double)numGoodSeasons;

                         // This subverts all erroneous coefficients due to rounding error //
                    if( (behaviorSource < .00000001 && behaviorSource > -.00000001) || (behaviorSurrogate < .00000001 &&
behaviorSurrogate > -.00000001) )
```

```
                        sourceData[seasonIndex(badSeason)][i] = "0";
                    else
                        sourceData[seasonIndex(badSeason)][i] =
""+(Double.parseDouble(surrogateData[seasonIndex(badSeason)][i]) * behaviorSource / behaviorSurrogate);

                }  catch( NumberFormatException e1 ) {
                    sourceData[seasonIndex(badSeason)][i] = surrogateData[seasonIndex(badSeason)][i];
                }
            }



            /**   Write sourceData to sourceFile   **/
            br1 = new BufferedReader( new FileReader( sourceFile ) );
            PrintStream ps = new PrintStream( new FileOutputStream( sourceFile+".tmp" ) );

            copyLines( br1,ps,40 );
            for( i = 0; i < 5; i++ )
            {
                ps.print("\t\t\t\t"+sourceData[i][0]+"\t\t");
                for( j = 1; j < numTokens; j++ )
                ps.print(sourceData[i][j]+"\t");

                ps.print("\n");
            }

            skipUntil( br1,"YYYY" );
            reprint( br1,ps );
            br1.close();
            ps.close();



            /**   Update sourceData with surrogated sourceData  **/
            File inputFile  = new File(sourceFile);
            File tempFile   = new File(sourceFile+".tmp");
            inputFile.delete();
            tempFile.renameTo(inputFile);


        } catch ( Exception e ) {
            //            println("Error in calculateSurrogateData!!");
            e.printStackTrace();
        }
    }


    public static void createWorksheet( String dprfiles[], String pagename, String reportTitle )
    {
        try {
        double  GN  = 0.0,
                TN  = 0.0,
                AN  = 0.0,
                P   = 0.0,
```

```
                MASS= 0.0;

        String s, s2;

        /** Parse dprfiles for calculations **/
        for( int i = 0; i < dprfiles.length; i++ )
        {
            /** Create BufferedReader object to read from each dpr file **/
            BufferedReader br = new BufferedReader( new FileReader( dprfiles[i] ) );
//            System.out.print( " F"+i+" " );
//            System.out.print( " F"+dprfiles[i]+" " );
            while( (s = br.readLine())!= null )
            {
                /** Get the header line for Annual info **/
                if( s.indexOf("Surface Station Number: ") != -1 )
                {
                    s = br.readLine();
                    StringTokenizer t = new StringTokenizer( s, "\t " );
                    /** Parse s to calculate variables **/
                    skip(t, 21);
                    GN    = Double.parseDouble(t.nextToken())+Double.parseDouble(t.nextToken());
                    MASS += Double.parseDouble(t.nextToken());
                    AN   += Double.parseDouble(t.nextToken())+Double.parseDouble(t.nextToken());
                    P    += Double.parseDouble(t.nextToken());
                    break; /* No further parsing is necessary */
                }
            }
            br.close();

        }
        TN = GN+AN;

        /** create PrintWriter to pagename (summary file) **/
        PrintStream pw = new PrintStream( new FileOutputStream(pagename) );
//        PrintStream pw = System.out;

            /*
             * PRINT THE TITLE STRING
             */
        pw.println(""+reportTitle);

        /** TODO: Copy ALL dpr summary data to pagename **/
        for( int i = 0; i < dprfiles.length; i++ )
        {
            /** Create BufferedReader object to read from each dpr file **/
            BufferedReader br = new BufferedReader( new FileReader( dprfiles[i] ) );
            while( (s = br.readLine())!= null )
            {
                /** Get the header line for Annual info **/
                if( s.indexOf("Diameter of Particulate Matter") != -1 )
                {
                    pw.println(s);  /** Print the Diameter Line **/
                    skip(br,4);
```

```
                    copyLines(br,pw,6);
//                  pw.println("");
                    break; /* No further parsing is necessary */
                }
            }
            br.close();
        }

        /** Write Summary Info! **/
        pw.print("\n");
        pw.println("\t\t\t\tx 0.15768");
        pw.println("\t\tGN\t"+GN+"\t"+GN*0.15768);
        pw.println("\t\tAN\t"+AN+"\t"+AN*0.15768);
        pw.println("\t\tTN\t"+TN+"\t"+TN*0.15768);
        pw.print("\n");
        pw.println("\t\tP\t"+P+"\t"+P*0.15768);
        pw.print("\n");
        pw.println("\t\tMass\t"+MASS+"\t"+MASS*0.15768);

        /** Close the pagename PrintWriter **/

        } catch( Exception e ) {
            System.out.println("Exception in SummarySummarizer:createWorksheet()\n\t"+e);
            e.printStackTrace();
        }
    }



      /**********************
       **  HOURLY METHODS  **
       **********************/
   public static void combineWorksheets_hourly( String[] pages, String[] titles, String workbookname )
    {
    try {
            /** create new workbook object **/
            HSSFWorkbook myWorkbook = new HSSFWorkbook();
            HSSFSheet currentSheet;

            HSSFFont  F_bold  = myWorkbook.createFont();
            F_bold.setBoldweight(HSSFFont.BOLDWEIGHT_BOLD );
            HSSFCellStyle S_bold  = myWorkbook.createCellStyle();
            S_bold.setFont(F_bold);

//            System.out.println("");

            for( int i = 0; i < pages.length; i++ )
            {
                /** create the BufferedReader for each page **/
                BufferedReader br=new BufferedReader(new FileReader(pages[i]));
                int row=0,col=0;
                String s;
```

```
                /** parse the report pages into workbook sheets **/
                if( titles[i].length() > 31 )
                    titles[i]=titles[i].substring(0,31);

//                System.out.println("title="+titles[i]+"  page="+pages[i]+"  workbookname="+workbookname);
                currentSheet = myWorkbook.createSheet(titles[i]);

                while( (s=br.readLine()) != null )
                {
                    s = replace(s,"\t"," @");
                    StringTokenizer t = new StringTokenizer( s, "@");
                    HSSFRow myRow = currentSheet.createRow((short)row++);
                    col=0;
                    while( t.hasMoreTokens() )
                    {
                        HSSFCell cell = myRow.createCell((short)col++);
                        switch( row )  {
                            case 1: case 3: case 4: case 5: case 131: case 132: case 258: case 259: case 385: case 386:
                                cell.setCellStyle(S_bold);
                        }

                        String ss = t.nextToken();
                        try {
                            cell.setCellValue( Double.parseDouble(ss) );
                        } catch( Exception e ) {
                            cell.setCellValue(ss);
                        }
//                        cell.setCellValue(t.nextToken());

                    }
                }
                br.close();
                System.out.print(".");
            }


            /** Write the workbook object out to (workbookname) file **/
            FileOutputStream fileOut = new FileOutputStream(workbookname);
            myWorkbook.write(fileOut);
            fileOut.close();
        } catch( Exception e ) {

            e.printStackTrace();
        }
    }

    public static void calculateSurrogateData_hourly( String sourceFile, String surrogateFile, String badSeason, String
goodSeasons )
    {
         // TODO: Convert this algorithm to hourly!!! //
  }


    public static void createWorksheet_hourly( String dprfiles[], String pagename, String reportTitle )
```

```
        // TODO:  Find Error in average calculation!! //
    {
        try {
        double  GN[]    = new double[125],
                TN[]    = new double[125],
                AN[]    = new double[125],
                P[]     = new double[125],
                MASS[]  = new double[125];
        String  time[]  = new String[125],
                season[]= new String[125];

        /** Initialize the Arrays **/
        int i,j;
        for( j = 0; j < 125; j++ )
        {
            GN[j]=-6999.0;
            TN[j]=-6999.0;
            AN[j]=-6999.0;
            P[j]=-6999.0;
            MASS[j]=-6999.0;
            time[j]="";
            season[j]="";
        }

        String HEADER_LINE="";
        String s, s2;

        /** Parse dprfiles for calculations **/
        for( i = 0; i < dprfiles.length; i++ )
        {
            /** Create BufferedReader object to read from each dpr file **/
            BufferedReader br = new BufferedReader( new FileReader( dprfiles[i] ) );
            while( (s = br.readLine())!= null )
            {
                /** Get the header line for Annual info **/
                if( s.indexOf("Surface Station Number") != -1 )
                {
                    for( j = 0; j < 125 ; j++ )
                    {
                        s = br.readLine();
                        StringTokenizer t = new StringTokenizer( s, "\t " );
                            /** Parse s to calculate variables **/
                        if( t.countTokens() < 31 ) {

                            time[j]   = t.nextToken();
                            season[j] = t.nextToken();
                        } else {
                            skip( t,22 );
                            GN[j]     = Double.parseDouble(t.nextToken()) + Double.parseDouble(t.nextToken());
                            MASS[j]   = Double.parseDouble(t.nextToken());
                            AN[j]     = Double.parseDouble(t.nextToken()) + Double.parseDouble(t.nextToken());
                            P[j]      = Double.parseDouble(t.nextToken());
                            time[j]   = t.nextToken();
```

```
                            season[j] = t.nextToken();
                        }
                    }
                    HEADER_LINE = br.readLine();
                    break; /* No further parsing is necessary */
                }
            }
            br.close();

        }
        for( j = 0; j < 125; j++ )
            if( GN[j] != -6999.0 )
                TN[j] = GN[j]+AN[j];

        /** create PrintWriter to pagename (summary file) **/
        PrintStream pw = new PrintStream( new FileOutputStream(pagename) );

         /* PRINT THE TITLE STRING  */
        pw.println(""+reportTitle);
        pw.println("\n\tOriginal Data");

        /** Copy ALL dpr summary data to pagename **/
        for( i = 0; i < dprfiles.length; i++ )
        {
            /** Create BufferedReader object to read from each dpr file **/
            BufferedReader br = new BufferedReader( new FileReader( dprfiles[i] ) );
            while( (s = br.readLine())!= null )
            {
                /** Get the header line for Annual info **/
                if( s.indexOf("Diameter of Particulate Matter") != -1 )
                {
                    pw.println(s);  /** Print the Diameter Line **/
                    skip(br,4);
                    pw.println(HEADER_LINE);
                    copyLines(br,pw,125);
                    break; /* No further parsing is necessary */
                }
            }
            br.close();
        }

        /** Close the pagename PrintWriter **/
        pw.close();

        } catch( Exception e ) {
            System.out.println("Exception in SummarySummarizer:createWorksheet()\n\t"+e);
            e.printStackTrace();
        }
    }
}
```

SheetImpoter.java –

Imports a sheet from one workbook to another.

```
package Batch2;
//package org.apache.poi.hssf;


import java.util.*;
import java.io.*;
import java.lang.*;
import org.apache.poi.hssf.model.*;
import org.apache.poi.hssf.usermodel.*;
import org.apache.poi.poifs.filesystem.*;

/**
 * SheetImporter - Class for processing
 *  the input Met files... This action could be broken
 *  into two separate operations, in the future.
 * @author Peter A Ruibal (ruibalp@csus.edu)
 * @author  Yatiraj Bhumkar (byatiraj@rediffmail.com)
 * @see Batch2.SummarySummarizer
 * @see Batch2.PropertiesViewer
 */
public class SheetImporter
{


        /** Imports a sheet from one workbook to another
         * @param source The source workbook filename
         * @param sheetname The sheet name to copy
         * @param destination The destination workbook filename
         */
    public static void importSheet( String source, String sheetname, String destination )  {
        try {

            POIFSFileSystem sfs = new POIFSFileSystem(new FileInputStream( source ));
            HSSFWorkbook swb = new HSSFWorkbook(sfs); /** Load the workbook from source **/


            POIFSFileSystem dfs = new POIFSFileSystem(new FileInputStream( destination ));
            HSSFWorkbook dwb = new HSSFWorkbook(dfs); /** Load the destination workbook **/

            if( dwb.getSheet(sheetname) != null )
            {
                System.out.println("\nDestination workbook already contains this sheet !!");
                return;
            }

            HSSFSheet copySheet = swb.getSheet( sheetname );
            HSSFSheet newSheet  = dwb.createSheet(sheetname);


            HSSFRow sRow,dRow;
            HSSFCell sCell,dCell;
```

```
                /** Copy the column width! **/
/*                for( short i = 0; i < 30; i++ )
                    newSheet.setColumnWidth( i,copySheet.getColumnWidth(i) );*/

            copySheet.setDefaultColumnWidth( newSheet.getDefaultColumnWidth() );
            copySheet.setDefaultRowHeight( newSheet.getDefaultRowHeight() );

                /** Iteratre through all rows! **/
            for( Iterator i = copySheet.rowIterator() ; i.hasNext() ; )
            {
                sRow = ((HSSFRow)i.next());

                dRow = newSheet.createRow( sRow.getRowNum() );
                dRow.setHeight(sRow.getHeight());

                    /** Iterate through all Columns! **/
                for( Iterator j = sRow.cellIterator() ; j.hasNext() ; )
                {
                    sCell = ((HSSFCell)j.next());
                    dCell = dRow.createCell( sCell.getCellNum() );

                    String sss="#ERR";
                    try {
                          /** Parse Formula Case! **/
                        if( sCell.getCellType() == HSSFCell.CELL_TYPE_FORMULA  )
                        {
                              /* Check for Sheet Reference, insert 's */
                            sss = sCell.getCellFormula();
                            if( sss.indexOf("!") != -1 )
                                sss = "\'"+replaceFirst(sCell.getCellFormula(),"!", "\'!");
//                                System.out.println("Formula: "+sss);
                            dCell.setCellFormula( sss );
                        } else if( sCell.getCellType() == HSSFCell.CELL_TYPE_NUMERIC )
                            dCell.setCellValue( sCell.getNumericCellValue() );
                        else if( sCell.getCellType() == HSSFCell.CELL_TYPE_STRING )
                            dCell.setCellValue( sCell.getStringCellValue() );
                    } catch( Exception e ) {
                      System.out.println("Exception on Formula: "+sss );
                      e.printStackTrace();
                    }

                        /* Handle Styles! */
                    HSSFCellStyle scs = sCell.getCellStyle(); // source style (from copy sheet)
                    HSSFCellStyle dcs;                        // destination style

                    short styleIndex = contains( dwb,scs );   // check new sheet for existing style
                    if( styleIndex == -1 ) {          // if it wasn't found,
                        dcs = dwb.createCellStyle();  //  create a new style ,
                        setStyle(scs,dcs);            //  initialize it
                        dCell.setCellStyle(dcs);      //  and set it
                    } else {                          // otherwise...
                        dCell.setCellStyle( dwb.getCellStyleAt(styleIndex) );
```

```
                    // There are still some formatting bugs //
                }

            }
        }

        // Write the output to a file
        FileOutputStream fileOut = new FileOutputStream( destination );
        dwb.write(fileOut);
        fileOut.close();

    } catch( Exception e ) {
        System.out.println("Exception in SheetImporter.importSheet():\n");
        e.printStackTrace();
    } /* catch( java.lang.OutOfMemoryError e2 ) {
        e2.printStackTrace();
    } */
}

private static short contains( HSSFWorkbook wb, HSSFCellStyle cs )
/**
 *  returns the index into the HSSFWorkbook where the matching HSSFStyle is located,
 *   -1 if not found
 */
{
    boolean found = false;
    for( short i = 0; i < wb.getNumCellStyles() ; i++ )
        if( equals(wb.getCellStyleAt(i),cs))
            return i;
    return -1;
}

private static void setStyle( HSSFCellStyle scs, HSSFCellStyle dcs )
{
    dcs.setAlignment( scs.getAlignment() );
    dcs.setBorderBottom( scs.getBorderBottom() );
    dcs.setBorderLeft( scs.getBorderLeft() );
    dcs.setBorderRight( scs.getBorderRight() );
    dcs.setBorderTop( scs.getBorderTop() );
    dcs.setDataFormat( scs.getDataFormat() );
    dcs.setBottomBorderColor( scs.getBottomBorderColor() );
    dcs.setTopBorderColor( scs.getTopBorderColor() );
    dcs.setLeftBorderColor( scs.getLeftBorderColor() );
    dcs.setRightBorderColor( scs.getRightBorderColor() );
    dcs.setFillBackgroundColor( scs.getFillBackgroundColor() );
    dcs.setFillForegroundColor( scs.getFillForegroundColor() );
    dcs.setFillPattern( scs.getFillPattern() );
    dcs.setHidden( scs.getHidden() );
    dcs.setIndention( scs.getIndention() );
    dcs.setLocked( scs.getLocked() );
    dcs.setRotation( scs.getRotation() );
    dcs.setVerticalAlignment( scs.getVerticalAlignment() );
    dcs.setWrapText( scs.getWrapText() );
```

```
    }

    private static boolean equals( HSSFCellStyle a, HSSFCellStyle b )
    {
        return (
            a.getAlignment() == b.getAlignment() &&
            a.getBorderBottom() == b.getBorderBottom() &&
            a.getBorderLeft() == b.getBorderLeft() &&
            a.getBorderTop() == b.getBorderTop() &&
            a.getBorderRight() == b.getBorderRight() &&
            a.getBottomBorderColor() == b.getBottomBorderColor() &&
            a.getLeftBorderColor() == b.getLeftBorderColor() &&
            a.getRightBorderColor() == b.getRightBorderColor() &&

            a.getDataFormat() == b.getDataFormat() &&

            a.getFillBackgroundColor() == b.getFillBackgroundColor() &&
            a.getFillForegroundColor() == b.getFillForegroundColor() &&
            a.getFillPattern() == b.getFillPattern() &&
                // getFontIndex //
            a.getHidden() == b.getHidden() &&
            a.getIndention() == b.getIndention() &&
            a.getHidden() == b.getHidden() &&
            a.getLocked() == b.getLocked() &&
            a.getRotation() == b.getRotation() &&
            a.getVerticalAlignment() == b.getVerticalAlignment() &&
            a.getWrapText() == b.getWrapText()
        );
    }




    private static String replaceFirst( String in, String change, String with )
    {
    int i;
        while( (i=in.indexOf(change)) != -1 )
        {
//            System.out.print(".");
            in = "" + in.substring(0,i) + with + in.substring(i+change.length());
            break;
        }
        return in;
    }



}
```