

# HydrotelPreAdapter

```
package nl.deltares.hydrotel;

import nl.wldelft.util.Box;
import nl.wldelft.util.FastDateFormat;
import nl.wldelft.util.FileUtils;
import nl.wldelft.util.LongArrayUtils;
import nl.wldelft.util.StringArrayUtils;
import nl.wldelft.util.TextUtils;
import nl.wldelft.util.TimeUnit;
import nl.wldelft.util.io.LineReader;
import nl.wldelft.util.io.LineWriter;
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import ucar.ma2.Array;
import ucar.nc2.Attribute;
import ucar.nc2.Dimension;
import ucar.nc2.NetcdfFile;
import ucar.nc2.Variable;
import ucar.nc2.units.DateUnit;

import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Path;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Properties;
import java.util.TimeZone;

public class HydrotelPreAdapter {
    private static final Logger log = Logger.getLogger(HydrotelPreAdapter.class);
    private static final String PROPERTIES = "properties";
    private static final String SORTIES_HYDROTEL = "sortiesHydrotel";
    private static final String SAUVEGARDE_ETAT_FIN_SIMULATION = "sauvegardeEtatFinSimulation";
    private static final String PR = "pr";
    private static final String STATION_ID = "station_id";

    private final File runFile;
    private Path runPath = null;
    private NetcdfFile netcdfRunFile = null;
    private String logLevel = "INFO";
    private static final String LOG_LEVEL = "log_level";
    private static final String WORK_DIR = "work_dir";
    private static final String END_TIME = "end_time";
    private static final String START_TIME = "start_time";
    private static final String INPUT_NETCDF_FILES_VARIABLE_NAME = "input_ncdf_files";
    private File workDirFile = null;
    private String[] inputTimeSeriesFiles = null;
    private float[] temperatureAdditifID = null;
    private float[] precipitationMultiplicatifID = null;
    private float[] precipitationAdditifID = null;
    private long[] correctionTimes = null;
    private File simulationFolder = null;
    private static final FastDateFormat YYYYMMDDHH = FastDateFormat.getInstance("yyyy/MM/dd HH", TimeZone.
getTimeZone("EST"), Locale.CANADA, null);
    private static final FastDateFormat YYYY_MM_DDHH = FastDateFormat.getInstance("yyyy-MM-dd HH", TimeZone.
getTimeZone("EST"), Locale.CANADA, null);
    private static final FastDateFormat DDMMYYYYKK = FastDateFormat.getInstance("dd/MM/yyyy kk", TimeZone.
getTimeZone("EST"), Locale.CANADA, null);
    private double startTime = 0.0;
```

```

private double endDateTime = 0.0;
private boolean saveStatesAtEnd = false;
private static final DecimalFormatSymbols DECIMAL_FORMAT_SYMBOLS = new DecimalFormatSymbols(Locale.CANADA);
private static final DecimalFormat DECIMAL_FORMAT = new DecimalFormat("0.0000",DECIMAL_FORMAT_SYMBOLS);

private HydrotelPreAdapter(File runFile) {
    this.runFile = runFile;
}

public static void main(String[] args) throws Exception {
    if (args.length != 1)
        throw new IllegalArgumentException("Specify either one argument that is the path to the run file,
or 4 arguments that are 'work dir', input file, output file and 'log HZ reeks'.");
    File runFile = new File(args[0]);
    if (!runFile.exists()) throw new Exception("Can not find run file specified as argument " + runFile);
    HydrotelPreAdapter adapter = new HydrotelPreAdapter(runFile);
    try {
        adapter.run();
        if (log.isInfoEnabled()) log.info("HydrotelAdapter run finished without exception");
    } catch (Exception e) {
        log.error(e.getMessage(), e);
        throw e;
    } finally {
        LogManager.shutdown();
    }
}

private void run() throws Exception {

    try {
        runPath = runFile.getParentFile().toPath();
        String absolutePath = runFile.getAbsolutePath();
        netcdfRunFile = NetcdfFile.open(absolutePath);
        readRunInfoFile();
    } finally {
        netcdfRunFile.close();
    }
    convertTimeSeries();
    editSimulationFile();
}

private void editSimulationFile() throws IOException {
    File simulationCsv = new File(simulationFolder, "simulation.csv");
    String[] simulationLines = FileUtils.readAllLines(simulationCsv);
    double writeStateTime = getWriteStateTime();
    for (int i = 1; i < simulationLines.length; i++) {
        String line = simulationLines[i];
        String[] splitLines = line.split(";");
        if (splitLines.length == 0) continue;
        String key = splitLines[0];
        if (splitLines.length == 1) splitLines = new String[]{key, null};
        switch (key) {
            case "DATE DEBUT":
                splitLines[1] = YYYY_MM_DDHH.format(startDate);
                break;
            case "DATE FIN":
                splitLines[1] = YYYY_MM_DDHH.format(endDate);
                break;
            case "ECRITURE ETAT FONTE NEIGE":
            case "ECRITURE ETAT BILAN VERTICAL":
            case "ECRITURE ETAT RUISSELEMENT SURFACE":
            case "ECRITURE ETAT ACHEMINEMENT RIVIERE":
                splitLines[1] = YYYY_MM_DDHH.format(writeStateTime);
                break;
            default:
                continue;
        }
        simulationLines[i] = TextUtils.join(splitLines, ';');
    }
    FileUtils.deleteIfExists(simulationCsv);
    FileUtils.writeText(simulationCsv, TextUtils.join(simulationLines, '\n'));
}

```

```

}

private double getWriteStateTime() {
    if (saveStatesAtEnd) return endDateTime;
    return startTime + TimeUnit.DAY_MILLIS;
}

private void readRunInfoFile() throws Exception {
    logLevel = netcdfRunFile.findVariable(LOG_LEVEL).readScalarString();
    getWorkDir();
    configureLogging();
    if (log.isDebugEnabled()) log.debug("HydrotelAdapter started");
    inputTimeSeriesFiles = getFilePaths(INPUT_NETCDF_FILES_VARIABLE_NAME);
    simulationFolder = new File(workDirFile, "simulation/simulation");
    readStartEndTime();
    handleProperties();
}

private void readStartEndTime() throws Exception {
    Variable startTimeVar = netcdfRunFile.findVariable(START_TIME);
    double relativeStartTime = startTimeVar.readScalarDouble();
    double relativeEndTime = netcdfRunFile.findVariable(END_TIME).readScalarDouble();
    Attribute units = startTimeVar.findAttribute("units");
    String tunitsString = units.getStringValue();
    if (log.isInfoEnabled()) log.info("INFO: Start time: " + relativeStartTime + ' ' + tunitsString);
    if (log.isInfoEnabled()) log.info("INFO: End time: " + relativeEndTime + ' ' + tunitsString);
    DateUnit referenceUnit = new DateUnit(units.getStringValue());
    long currentReferenceTime = referenceUnit.getDateOrigin().getTime();
    ucar.nc2.units.TimeUnit timeUnit = referenceUnit.getTimeUnit();
    startTime = timeUnit.getValueInSeconds(relativeStartTime) * 1000 + currentReferenceTime;
    endTime = timeUnit.getValueInSeconds(relativeEndTime) * 1000 + currentReferenceTime;
}

private void getWorkDir() throws IOException {
    String relativeWorkDirString = String.valueOf((char[])) netcdfRunFile.findVariable(WORK_DIR).read().
copyToNDJavaArray()).trim();
    String workDirString = getWorkDirString(relativeWorkDirString);
    workDirFile = new File(workDirString);
    if (log.isInfoEnabled()) log.info("Work dir: " + workDirString);
    if (!workDirFile.exists()) throw new RuntimeException("Work dir not found: " + workDirString);
}

private String getWorkDirString(String relativeWorkDirString) {
    if (TextUtils.equals(relativeWorkDirString, ".") || relativeWorkDirString.equals(".")) return runPath.toString();
    return runPath.resolve(new File(relativeWorkDirString).toPath()).toString();
}

private void handleProperties() throws Exception {
    Variable var = netcdfRunFile.findVariable(PROPERTIES);
    String sortiesHydrotel = getPropertyStringValue(var, SORTIES_HYDROTEL);
    convertOutputSettings(sortiesHydrotel);
    String saveStateAtEndStringValue = getPropertyStringValue(var, SAUVEGARDE_ETAT_FIN_SIMULATION);
    if (saveStateAtEndStringValue != null) saveStatesAtEnd = Boolean.valueOf(saveStateAtEndStringValue);
}

private void convertOutputSettings(String sortiesHydrotel) throws IOException {
    boolean debitAvalOnly = false;
    String[] outputValuesHydrotel = new String[0];
    if (sortiesHydrotel == null) {
        debitAvalOnly = true;
    } else {
        outputValuesHydrotel = sortiesHydrotel.split(";");
        if (outputValuesHydrotel.length == 1 && outputValuesHydrotel[0].equals("DEBIT_AVAL")) {
            debitAvalOnly = true;
        }
    }
    File outputCsv = new File(simulationFolder, "output.csv");
    if (debitAvalOnly) {
        File exutoireCsv = new File(simulationFolder, "output_exutoire.csv");
        FileUtils.copy(exutoireCsv, outputCsv);
    } else {
}

```

```

        File bassinCsv = new File(simulationFolder, "output_bassin.csv");
        String[] bassinLines = FileUtils.readAllLines(bassinCsv);
        for (int i = 1; i < bassinLines.length; i++) {
            String line = bassinLines[i];
            String[] splitLines = line.split(";");
            if (StringArrayUtils.contains(outputValuesHydrotel, splitLines[0])) {
                splitLines[1] = "1";
            }
            bassinLines[i] = TextUtils.join(splitLines, ';');
        }
        FileUtils.writeText(outputCsv, TextUtils.join(bassinLines, '\n'));
    }
}

private String[] getFilePaths(String variableName) throws IOException {
    Variable inputVar = netcdfRunFile.findVariable(variableName);
    Array inputArray = inputVar.read();
    Object inputFilesNDArrayObject = inputArray.copyToNDJavaArray();
    char[][] inputFilesChar = (char[][][]) inputFilesNDArrayObject;

    List<String> inputFilesList = new ArrayList<>();

    for (char[] chArray : inputFilesChar) {
        File inputFileRelativePath = new File(String.valueOf(chArray).trim());
        String absoluteFilePath = runPath.resolve(inputFileRelativePath.toPath()).toString();
        inputFilesList.add(absoluteFilePath);
    }

    String[] inputFiles = new String[inputFilesList.size()];
    inputFilesList.toArray(inputFiles);
    return inputFiles;
}

private static List<String> getStringsFromNetcdfVariable(NetcdfFile netcdfFile, String variableName) throws
IOException {
    Variable inputVar = netcdfFile.findVariable(variableName);
    Array inputArray = inputVar.read();
    Object inputFilesNDArrayObject = inputArray.copyToNDJavaArray();
    char[][] inputFilesChar = (char[][][]) inputFilesNDArrayObject;

    List<String> stationIds = new ArrayList<>();

    for (char[] chArray : inputFilesChar) {
        String stationId = String.valueOf(chArray).trim();
        stationIds.add(stationId);
    }

    return stationIds;
}

private void configureLogging() {
    File logFile = new File(workDirFile, "hydrotel_pre_adapter.log");
    if (logFile.exists()) {
        //noinspection ResultOfMethodCallIgnored
        logFile.delete();
    }
    Properties props = new Properties();
    props.setProperty("log4j.rootLogger", logLevel + ", hydrotel");
    props.setProperty("log4j.appenders.hydrotel.File", logFile.getAbsolutePath());
    props.setProperty("log4j.appenders.hydrotel", "org.apache.log4j.RollingFileAppender");
    props.setProperty("log4j.appenders.hydrotel.layout", "org.apache.log4j.PatternLayout");
    props.setProperty("log4j.appenders.hydrotel.layout.ConversionPattern", "%p - %m \n");
    LogManager.resetConfiguration();
    PropertyConfigurator.configure(props);
    if (log.isDebugEnabled()) log.debug("Debug log level enabled");
    if (log.isInfoEnabled()) log.info("Info log level enabled");
}

private void convertTimeSeries() throws Exception {
    String meteoFile = null;
    for (String inputTimeSeriesFile : inputTimeSeriesFiles) {

```

```

        if (inputTimeSeriesFile.endsWith("meteo.nc")) {
            meteoFile = inputTimeSeriesFile;
        } else if (inputTimeSeriesFile.endsWith("corrections.nc")) {
            convertCorrections(inputTimeSeriesFile);
        } else if (inputTimeSeriesFile.endsWith("debit.nc")) {
            convertHydro(inputTimeSeriesFile);
        }
    }
    convertMeteo(meteoFile);
}

private void convertHydro(String inputTimeSeriesFile) throws IOException {
    File hydroDir = new File(workDirFile, "hydro");
    File stationsSth = new File(hydroDir, "station.sth");
    List<String> requiredStations = getRequiredStations(stationsSth);
    NetcdfFile netcdfFile = null;
    long[] times;
    float[][] floats;
    List<String> stationIds;
    try {
        netcdfFile = NetcdfFile.open(inputTimeSeriesFile);
        Variable timeVar = netcdfFile.findVariable("time");
        times = readTimes(timeVar);
        boolean useExtraEnsembleDimension = useExtraEnsembleDimension(netcdfFile);
        floats = read2DArray(netcdfFile, "Q_obs", useExtraEnsembleDimension);
        stationIds = getStringsFromNetcdfVariable(netcdfFile, STATION_ID);
    } finally {
        if (netcdfFile != null) netcdfFile.close();
    }
    writeHydroFiles(hydroDir, stationsSth, requiredStations, times, floats, stationIds, netcdfFile);
}

private static void writeHydroFiles(File hydroDir, File stationsSth, List<String> requiredStations, long[]
times, float[][] floats, List<String> stationIds, NetcdfFile netcdfFile) throws IOException {
    for (int i = 0; i < stationIds.size(); i++) {
        String id = stationIds.get(i);
        if (!requiredStations.contains(id)) {
            log.warn("Data exported for station " + id + " but not present in " + stationsSth);
            continue;
        }
        requiredStations.remove(id);
        writeHydroStationFile(hydroDir, times, floats, i, id);
    }
    // Log errors for stations not removed in loop above
    for (String requiredStation : requiredStations) {
        log.error("No data for station " + requiredStation + " found in " + netcdfFile.getLocation());
    }
}

private static void writeHydroStationFile(File hydroDir, long[] times, float[][] floats, int i, String id)
throws IOException {
    File hydroStationFile = new File(hydroDir, id + ".hyd");
    try (LineWriter lineWriter = new LineWriter(hydroStationFile, Charset.defaultCharset())) {
        lineWriter.writeLine("1 3");
        lineWriter.writeLine("");
        for (int j = 0; j < times.length; j++) {
            String dateTime = DDMMYYYYKK.format(times[j]);
            String value = TextUtils.padLeft(DECIMAL_FORMAT.format(floats[j][i]), 8);
            lineWriter.writeLine(dateTime + ' ' + value);
        }
    }
}

private static List<String> getRequiredStations(File stationsSth) throws IOException {
    List<String> requiredStations = new ArrayList<>();
    try (LineReader lineReader = new LineReader(stationsSth, Charset.defaultCharset())) {
        lineReader.skipLines(3);
        String line = lineReader.readLine();
        while (line != null) {
            String[] columns = line.split(" ");

```

```

        requiredStations.add(columns[0]);
        line = lineReader.readLine();
    }
}
return requiredStations;
}

private void convertCorrections(String inputTimeSeriesFile) throws IOException {
    NetcdfFile netcdfFile = null;
    try {
        netcdfFile = NetcdfFile.open(inputTimeSeriesFile);
        Variable timeVar = netcdfFile.findVariable("time");
        correctionTimes = readTimes(timeVar);
        float[] eenAdditif1D = read1DArrayFloat(netcdfFile, "eenAdditif");
        float[] reserveEauSolMultiplicatif1D = read1DArrayFloat(netcdfFile, "reserveEauSolMultiplicatif");

        File correctionCsv = new File(simulationFolder, "correction.csv");
        String correctionText = FileUtils.readText(correctionCsv);
        File correctionTmp = new File(simulationFolder, "correction_tmp.csv");
        try (LineWriter lineWriter = new LineWriter(correctionTmp, Charset.defaultCharset())) {
            lineWriter.write(correctionText);
            lineWriter.newLine("");
            writeEenAdditiveCorrection(eenAdditif1D, lineWriter, DECIMAL_FORMAT);
            writeReserveEauSolMultiplicatifCorrection(reserveEauSolMultiplicatif1D, lineWriter,
DECIMAL_FORMAT);
        }
        FileUtils.deleteIfExists(correctionCsv);
        FileUtils.copy(correctionTmp, correctionCsv);
        FileUtils.deleteIfExists(correctionTmp);

        precipitationAdditif1D = read1DArrayFloat(netcdfFile, "precipitationAdditif");
        precipitationMultiplicatif1D = read1DArrayFloat(netcdfFile, "precipitationMultiplicatif");
        temperatureAdditif1D = read1DArrayFloat(netcdfFile, "temperatureAdditif");
    } finally {
        if (netcdfFile != null) netcdfFile.close();
    }
}

private void writeEenAdditiveCorrection(float[] values, LineWriter lineWriter, DecimalFormat decimalFormat)
throws IOException {
    String[] strings = new String[8];
    for (int k = 0; k < correctionTimes.length; k++) {
        float value = values[k];
        if (value == 0.0f) continue;
        String dateTimeString = YYYYMMDDHH.format(correctionTimes[k]);
        if (Float.isNaN(value)) throw new RuntimeException("NaN value found for eenAdditif at " +
dateTimeString);
        strings[0] = "1";
        strings[1] = dateTimeString;
        strings[2] = dateTimeString;
        strings[3] = "5";
        strings[4] = decimalFormat.format(value);
        strings[5] = "1";
        strings[6] = "0";
        strings[7] = "tous";
        String line = TextUtils.join(strings, ';');
        lineWriter.newLine(line);
    }
}

private void writeReserveEauSolMultiplicatifCorrection(float[] values, LineWriter lineWriter, DecimalFormat
decimalFormat) throws IOException {
    String[] strings = new String[8];
    for (int k = 0; k < correctionTimes.length; k++) {
        float value = values[k];
        if (value == 1.0f) continue;
        String dateTimeString = YYYYMMDDHH.format(correctionTimes[k]);
        if (Float.isNaN(value))
            throw new RuntimeException("NaN value found for reserveEauSolMultiplicatif at " +
dateTimeString);
        strings[0] = "1";

```

```

        strings[1] = dateDateTimeString;
        strings[2] = dateDateTimeString;
        strings[3] = "4";
        strings[4] = "0";
        strings[5] = decimalFormat.format(value);
        strings[6] = "0";
        strings[7] = "tous";
        String line = TextUtils.join(strings, ';');
        lineWriter.writeLine(line);
    }
}

private static double[] read1DArrayDouble(NetcdfFile netcdfFile, String varName) throws IOException {
    Variable eenAdditif = netcdfFile.findVariable(varName);
    Array eenAdditifArray = eenAdditif.read();
    return (double[]) eenAdditifArray.copyTo1DJavaArray();
}

private static float[] read1DArrayFloat(NetcdfFile netcdfFile, String varName) throws IOException {
    Variable variable = netcdfFile.findVariable(varName);
    Array array = variable.read();
    return (float[]) array.copyTo1DJavaArray();
}

private void convertMeteo(String inputTimeSeriesFile) throws IOException {
    File meteoDir = new File(workDirFile, "meteo");
    Map<Box<Float, Float>, String> requiredXYCoords = getRequiredCoords(meteoDir);

    NetcdfFile netcdfFile = null;
    try {
        netcdfFile = NetcdfFile.open(inputTimeSeriesFile);
        double[] x = read1DArrayDouble(netcdfFile, "x");
        double[] y = read1DArrayDouble(netcdfFile, "y");
        boolean useExtraEnsembleDimension = useExtraEnsembleDimension(netcdfFile);
        Variable timeVar = netcdfFile.findVariable("time");
        long[] times = readTimes(timeVar);
        //noinspection ResultOfMethodCallIgnored
        meteoDir.mkdirs();
        float[][][] prFloats = read3DArray(netcdfFile, PR, useExtraEnsembleDimension);
        float[][][] tasMinFloats = read3DArray(netcdfFile, "tasmin", useExtraEnsembleDimension);
        float[][][] tasMaxFloats = read3DArray(netcdfFile, "tasmax", useExtraEnsembleDimension);
        writeRequiredMeteoFiles(meteoDir, requiredXYCoords, x, y, times, DDMMYYYYKK, prFloats,
tasMinFloats, tasMaxFloats);
    } finally {
        if (netcdfFile != null) netcdfFile.close();
    }
}

private static boolean useExtraEnsembleDimension(NetcdfFile netcdfFile) {
    Dimension realization = netcdfFile.findDimension("realization");
    if (realization == null) return false;
    if (realization.getLength() > 1) throw new RuntimeException("Multiple ensembles at once not supported");
    return true;
}

private void writeRequiredMeteoFiles(File meteoDir, Map<Box<Float, Float>, String> requiredXYCoords, double
[] x, double[] y, long[] times, FastDateFormat dateFormat, float[][][] prFloats, float[][][] tasMinFloats, float
[][][] tasMaxFloats) throws IOException {
    for (int i = 0; i < x.length; i++) {
        float xCoord = (float) x[i];
        for (int j = 0; j < y.length; j++) {
            float yCoord = (float) y[j];
            Box<Float, Float> foundCoords = new Box<>(xCoord, yCoord);
            String fileName = requiredXYCoords.get(foundCoords);
            if (fileName == null) continue;
            File meteoFile = new File(meteoDir, fileName + ".met");
            writeMeteoFile(times, dateFormat, prFloats, tasMinFloats, tasMaxFloats, i, j, meteoFile);
        }
    }
}

```

```

    private void writeMeteoFile(long[] times, FastDateFormat dateFormat, float[][][] prFloats, float[][][] tasMinFloats, float[][][] tasMaxFloats, int i, int j, File meteoFile) throws IOException {
        try (LineWriter lineWriter = new LineWriter(meteoFile, Charset.defaultCharset())) {
            lineWriter.writeLine("1 3");
            String[] strings = new String[5];
            for (int k = 0; k < times.length; k++) {
                assert correctionTimes[k] == times[k];
                String[] dateStrings = dateFormat.format(times[k]).split(" ");
                strings[0] = dateStrings[0];
                strings[1] = dateStrings[1];
                strings[2] = TextUtils.padLeft(String.valueOf(tasMaxFloats[k][j][i] + temperatureAdditif1D[k]),
5);
                strings[3] = TextUtils.padLeft(String.valueOf(tasMinFloats[k][j][i] + temperatureAdditif1D[k]),
5);
                strings[4] = String.valueOf(prFloats[k][j][i] * precipitationMultiplicatif1D[k] +
precipitationAdditif1D[k]);
                String line = TextUtils.join(strings, '_').replaceAll("_", " ");
                lineWriter.writeLine(line);
            }
        }
    }

    private static Map<Box<Float, Float>, String> getRequiredCoords(File meteoDir) throws IOException {
        File stationStm = new File(meteoDir, "station.stm");

        Map<Box<Float, Float>, String> requiredXYCoords = new HashMap<>();
        try (LineReader lineReader = new LineReader(stationStm, Charset.defaultCharset())) {
            lineReader.skipLines(3);
            String line = lineReader.readLine();
            while (line != null) {
                String[] columns = line.split(" ");
                float x = Float.valueOf(columns[1]);
                float y = Float.valueOf(columns[2]);
                Box<Float, Float> coordBox = new Box<>(x, y);
                requiredXYCoords.put(coordBox, columns[0]);
                line = lineReader.readLine();
            }
        }
        return requiredXYCoords;
    }

    private static float[][][] read3DArray(NetcdfFile netcdfFile, String varName, boolean useExtraEnsembleDimension) throws IOException {
        Variable variable = netcdfFile.findVariable(varName);
        Array values = variable.read();
        if (!useExtraEnsembleDimension) return (float[][][]) values.copyToNDJavaArray();
        int[] shape = variable.getShape();
        if (shape.length == 3) {
            log.warn("Not all variables contain an ensemble dimension");
            return (float[][][]) values.copyToNDJavaArray();
        }
        float[][][] floats4D = (float[][][]) values.copyToNDJavaArray();
        float[][][] floats3D = new float[floats4D.length][][];
        for (int i = 0; i < floats4D.length; i++) {
            floats3D[i] = floats4D[i][0];
        }
        return floats3D;
    }

    @SuppressWarnings("SameParameterValue")
    private static float[][] read2DArray(NetcdfFile netcdfFile, String varName, boolean useExtraEnsembleDimension) throws IOException {
        Variable variable = netcdfFile.findVariable(varName);
        Array values = variable.read();
        if (!useExtraEnsembleDimension) return (float[][]) values.copyToNDJavaArray();
        int[] shape = variable.getShape();
        if (shape.length == 2) {
            log.warn("Not all variables contain an ensemble dimension");
            return (float[][]) values.copyToNDJavaArray();
        }
    }
}

```

```

        }

        float[][][] floats3D = (float[][][]) values.copyToNDJavaArray();
        float[][] floats2D = new float[floats3D.length][];
        for (int i = 0; i < floats2D.length; i++) {
            floats2D[i] = floats3D[i][0];
        }
        return floats2D;
    }

    private static String getPropertyStringValue(Variable propertiesVariable, String propertyName) throws
Exception {
    if (propertiesVariable == null) return null;
    Attribute attribute = propertiesVariable.findAttribute(propertyName);
    if (attribute == null) return null;

    String value = attribute.getStringValue();
    if (value == null || value.trim().isEmpty()) {
        throw new Exception("Property '" + propertyName + "' in netcdf run file is empty or is not of type
String.");
    }

    String trimmedValue = value.trim();
    if (log.isDebugEnabled())
        log.debug("Read property " + propertyName + " with value '" + trimmedValue + "' from netcdf run
file.");
    return trimmedValue;
}

private static long[] readTimes(Variable timeVariable) throws IOException {
    if (timeVariable == null) return LongArrayUtils.EMPTY_ARRAY;

    //read times.
    Array timeArray = timeVariable.read();
    double[] times = (double[]) timeArray.get1DJavaArray(Double.class);

    //convert times.
    long[] convertedTimes = new long[times.length];
    DateUnit dateUnit = readTimeUnit(timeVariable);
    if (dateUnit == null)
        throw new IOException("Invalid date time unit string has been coded in the file: '" + timeVariable.
getUnitsString() + "'. Unit string should be for example: 'seconds since 2012-01-30 00:00:00'");
    for (int i = 0; i < times.length; i++) {
        Date date = dateUnit.makeDate(times[i]);
        if (date == null) throw new RuntimeException("Invalid time the file: '" + times[i]);
        convertedTimes[i] = date.getTime();
    }
    return convertedTimes;
}

private static DateUnit readTimeUnit(Variable timeVariable) {
    try {
        String unitString = timeVariable.getUnitsString();

        //if present, replace . by : in the timeZone specification in the unitString,
        //e.g. change "days since 2011-09-19 06:0:0.0 0.00" to "days since 2011-09-19 06:0:0.0 0:00".
        //This is a workaround implemented for FEWS-6544.
        String[] parts = unitString.split("\s+");
        if (parts.length > 0 && parts[parts.length - 1].matches(".?\\d{1,2}\\.\\d{2}"))
            //if a timeZone specification is present, then it is always the last part, see
            //http://cf-pcmdi.llnl.gov/documents/cf-conventions/1.5/cf-conventions.html#time-coordinate
            parts[parts.length - 1] = parts[parts.length - 1].replaceFirst("\\.", ":");

        StringBuilder buffer = new StringBuilder(parts[0]);
        for (int n = 1; n < parts.length; n++) {
            buffer.append(' ').append(parts[n]);
        }
        unitString = buffer.toString();
    }
}

```

```
        return new DateUnit(unitString);
    } catch (Exception e) {
        //if the given variable does not have a unit of time.
        return null;
    }
}
```