```
package com.analyticsanvil.custominputformat;

import java.io.IOException;
import java.io.DataInputStream;
import java.io.InputStream;
import java.util.Arrays;

import org.apache.commons.io.IOUtils;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.CompressionCodecFactory;
import org.apache.hadoop.mapred.InputSplit;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.FileSplit;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.LineRecordReader;
import org.apache.hadoop.mapred.Reporter;

public class CustomTextRecordReader implements RecordReader<LongWritable, BytesWritable> {

    private FSDataInputStream fsInput;
    private final LongWritable key = new LongWritable();
    private final BytesWritable value = new BytesWritable();
    private long start, end, pos;
    private boolean fileProcessed = false;
    private DataInputStream in;
    private InputStream dcin;

    private Text text;
    private byte[] textBytes;
    byte[] binaryData;
    int i = 0;
    int from = 0;
    LineRecordReader reader;

    CustomTextRecordReader(InputSplit genericSplit, JobConf job, Reporter reporter) throws IOException {
        FileSplit split = (FileSplit) genericSplit;
        Path path = split.getPath();
        FileSystem fs = path.getFileSystem(job);
        fsInput = fs.open(path);

        // Decompress file if it's compressed (e.g. GZIP, BZIP2)
        CompressionCodecFactory compressionCodecs = new CompressionCodecFactory(job);
        final CompressionCodec codec = compressionCodecs.getCodec(path);

        if (codec != null) {
            dcin = codec.createInputStream(fsInput);
            in = new DataInputStream(dcin);
        } else {
```

```
            dcin = null;
            in = fsInput;
        }

        start = split.getStart();
        end = start + split.getLength();

        // Copy decompressed input to a byte array
        textBytes = IOUtils.toByteArray(in);

        // Transform the file using the custom class
        binaryData = ReadASCIIGridFile.translateInput(textBytes);

    }

    @Override
    public boolean next(LongWritable key, BytesWritable value) throws IOException {

        // Read from the output byte array until a newline is found
        while (i < binaryData.length) {
            while (binaryData[i] != '\n') {
                i++;
            }

            // Set the key to the row number (arbitrary - key is not used by Hive)
            key.set(i);

            // Set the value to the full row read from this part of the output byte array
            value.set(binaryData, from, i - from);
            from = i;

            // Increment rowcount
            i++;

            // There is still more data...
            return true;
        }
        // There is no more data.
        this.fileProcessed = true;
        return false;

    }

    // Generate new key object
    @Override
    public LongWritable createKey() {
        return new LongWritable();
    }

    // Generate new value object
    @Override
    public BytesWritable createValue() {
        return new BytesWritable();
    }
```

```
    // Return position in file
    @Override
    public long getPos() throws IOException {
        return pos;
    }

    // Close input streams (raw and uncompressed)
    @Override
    public void close() throws IOException {
        in.close();
        dcin.close();
    }

    // Progress for this file is 1 when finished processing, 0 otherwise
    @Override
    public float getProgress() throws IOException {
        if (fileProcessed) {
            return 1.0f;
        } else {
            return 0.0f;
        }

    }
}
```