# Reading & Writing Text Files

A common operation, especially in simple data utilities, is reading and writing formatted data to and from simple text files. Sometimes these file are actual CSV format, and other times they are, or could be, more complex text formats. Vanilla has two components that simplify the reading and writing of these simple text-based data files.

## TextFileWriter

The `TextFileWriter` is created with an output file and a `LineFormatter`, which is used to write out the text lines based on t the incoming data. The `LineFormatter` interface provides two hooks for writing out the text data:

```
interface LineFormatter {

    String formatComment(String text)

    String formatLine(Object object)
}
```

One method for writing comments to the file in a format-appropriate manner, and the other for writing the formatted data of an input object.

A default implementation of the `LineFormatter` interface is provided as the `CommaSeparatedLineFormatter` which writes out data as comma-separated values of a collection representing the row (line) of data. The `toString()` method of each element is used to write the individual elements to the line.

An example using the `TextFileWriter` with the provided `CommaSeparatedLineFormatter` would be:

```
TextFileWriter writer = new TextFileWriter(
    lineFormatter: new CommaSeparatedLineFormatter(),
    filePath: new File('/some/file/out.txt')
)

writer.writeComment('This is a header')
writer.write(['a','b','c'])
writer.write(['d','e','f'] as Object[])
writer.writeComment('This is a footer')
```

## TextFileReader

Reading from simple text-based data formats is also quite common, and made easier with the

TextFileReader class. A TextFileReader instance takes a File, URI, or URL file location and a LineParser implementation instance. The LineParser will be used to process each line of text from the file and convert it to Strings or some other configured data type.

The LineParser interface provides three methods:

```
interface LineParser {

    boolean parsable(String line)

    Object[] parseLine(String line)

    Object parseItem(Object item, int index)
}
```

One method to determine if a line should be parsed, one two parse the line and a third to parse the individual line items - it is this third method that allows for the individual items of a line to converted into any desired type.

The default provided implementation of the LineParser interface is the CommaSeparatedLineParser which allows for configuration of the line item converters as mappings of row index number to the conversion closure.

An example of the TextFileReader with the CommaSeparatedLineParser would be:

```
TextFileReader reader = new TextFileReader(
    filePath: new File('/text-file.txt'),
    lineParser: new CommaSeparatedLineParser(
        (1): { v -> (v as Long) * 2 }
    ),
    firstLine: 2
)

def lines = []

reader.eachLine { Object[] data ->
    lines << data
}
```

These two text operation components are not really intended for high-performance applications, but more for quick utilities or sideline data loading operations.