## Documentation

Usage documentation is available in the readme file in the project. All code is documented via JSDocs format should be sufficient for a developer to understand what’s going on.

## Architecture Overview (Brief)

### Language Selection

For this project I choose NodeJS for 2 reasons. First, was my familiarity with the language, and the tools (modules) available for development and unit testing. Second, was the NodeJS streaming interface. This system needs to be able to ingest large files, and streaming allows each chunk to be parsed and resources to be released, before proceeding to the next chunk.

### Functional Programming

This project is essentially a simple ETL process, which lends itself well to a pattern of read data, pipe data to a transformation pipeline, pipe data to an output destination.

### Code Overview

Code is structured around the concept of extract, transform and load. Upon calling ingest(sourceFile, destFile, config); function. A function pipe (flow) is created with the rules specified by the passed in config (configs are stored as flat files in lib/ingestionConfig). That wired up pipe then acts as the transform step. Finally data is streamed from the sourceFile, parsed from CSV to object, then streamed row by row to the transform pipe, and finally loaded (to a flat file for this example code) to the destFile.

## Assumptions / Simplifications

* Hyphens are allowed in Product Number. The regex example doesn’t show that, however the example file input had it, and a string (as per the schema) can support it
* Loading into a DB (since none was specified etc) will be handled by an external process, or will be developed in a later iteration.
* Config files will be in the correct format (no application verification) and we’re simplified to be included inside the code base itself.
* A process will ingest error logs from console.error()
* Process Starts / End / Progress logging was scoped out.

## Next Steps

* First thing is the data needs to be loaded into some sort of a datastore. Flat files are great for demo’s, not so good for production.
* Second (equally as important), is scheduling and logging. This process would need to modified to provide hooks for external monitoring / logging, or if it going to be used as a stand alone app, would need to conform to the existing monitoring and logging infrastructure
* Third, I would work on the generation of the config files. They need to be validated, preferably generated by a tool of some variety, and preferably moved into a database (MongoDB was what I had in mind when I generated these). They could be kept in flat files, but would need to be moved out of the code base and included back in (S3), or dependency injected upstream.