The module is to be written in Ruby. We have chosen this language for a number of reasons, above simply being most comfortable with the language.

Firstly, the parsing of information is made much easier using Ruby, making it a much more practical language for what we need. It is easily achieved using the regular expression tools that Ruby provides. Secondly, it uses principles of Object Oriented Programming – one which the developers of the module are strongly familiar with. These advantages are all encompassed in another functionality that Ruby provides. The files provided for input are all of a comma-separated value format. Ruby furthers the ease of using these files by providing classes to read in CSV files. Each record is then stored as an object, with characteristics stored as per its description within the CSV file, further ensuring a simplified program to achieve the aim.

Lastly, it provides a couple feasible methods to develop and execute our module – which are applicable for other languages, of course. However, in our considerations, we also considered conversion of scripts to actual usable applications. However, we must first consider which operating system we should use.

If we were to use the Windows Operating System, the available methods for running our module would be:

1. Use the command prompt - fairly simple to use and effective, can run the module on the command prompt and direct input and output files to the module
2. Use an IDE - very easy to use, can edit code while running, can easily monitor input and output files. IDEs include:
   1. EditRocket
   2. RubyMine
3. Creating an executable (‘.exe’) file using a gem such as “Ocra”

We can choose to use either EditRocket or RubyMine IDE, which would make editing and testing the module very simple and effective. We could also visually control the input and output through the IDE. Both IDEs are compatible with multiple operating systems, meaning we could operate the module on both.

For options 1-2 however, one setback with using the Windows OS is that Ruby will need to be installed before use. Option 3, however, is a feasible option that does not have this setback, and has the full capacity to run as a standalone module.

If we were to use Mac OS X operating system, the available methods for setting up an environment, as well as running our module would be:

1. The Mac terminal- fairly simple to use and effective, can run the module on the terminal and direct input and output files to the module
2. Use an IDE - as mentioned above

The Mac OS X has Ruby pre-installed, therefore less processes to get the module running.

If we plan to use just the command prompt/ terminal we would need to run the module from it, and direct out input and output to it manually in our command. This is simple although during testing would be hard to make quick changes. The Mac OS X terminal would be the better option as Ruby is pre-installed, making it easier to get the environment set up.

The decision follows that, using the command prompt/terminal wouldn’t require the installation of third-party IDE’s, and hence would require less training for the developers. However, in using an IDE, which requires an initial installation, we will allow us to handle editing, input/output files and running the module in one window. Our final decision is as per Section 2, whereby there will be two different types of modules to account for Mac OSX, as well as Windows and Linux, separately.