**Purpose:**

The purpose of the software is to facilitate the manipulation of SFM files so that they can be imported into FieldWorks.

It will do that by reporting on the current state of the SFM file and allow the user to decide which changes to make. Then the program will save a modified copy of the SFM file.

The program should also be able to read and write FieldWorks importing MAP files. This will save the user a lot of duplicated effort. If they have a map file, then this program can read it to know which markers correspond to which fields in FLEx. If it doesn’t exist then it can be started in this program and save the duplication of effort when importing into FLEx.

**Scope:**

The program is expected to read and write SFM and CSV files for the linguistic data, and to read and write MAP files for importing into FLEx. It should be able to convert data between SFM and various CSV styles.

**System Overview**

**References**<http://www.angelfire.com/planet/linguisticsisfun/MDFDocumentation.zip>

http://downloads.sil.org/legacy/shoebox/MDF\_2000.pdf

**Definitions**SFM files are dictionary files created by Toolbox or Shoebox. They are often based on the Multi-Dictionary Format specification, however each linguist may have used their own variations.

**Tasks that the software should perform:**

1. Check that the file is Unicode throughout.
2. Report on the markers used.
3. Look at the number of occurrences of characters per marker. Report on rare characters for a given marker so that the user can see whether these are errors in the data.
4. Report on the contents of markers that have a limited set of contents. These are “Range Sets” in Toolbox, and “List fields” in FLEx.
5. Allow the user to modify markers by replacing them with another. E.g. in the case where two markers are used to represent the same thing. \xfr => \xn for example. Or in the case where we want to change all the subentries into main entries \se => \lx
6. Allow the user to modify data in a given marker. E.g. in the case where two parts of speech abbreviations are used to represent the same thing. \ps prép => \ps prep
7. Convert between CSV files and SFM. The CSVs file might have markers as column headings, or the markers may be included with each field, note that in this second form the data is not lined up in columns and empty cells are not required. The program should be able to read and write both forms of CSV file. The difficulty problem with the standard csv output (with data lined up in columns) is that different entries have varying numbers of repeating fields. So if one entry has 5 senses there must be columns for 5 senses though most records in the csv will only have one.
8. Check that certain markers occur in pairs. For example \lf should always have a corresponding \lv. Though \lv could occur alone there’s no point marking something as a synonym without indicating which word is the synonym. Note that the \lf Syn = word format is recommended by MDF. The code should cope with both formats and in both cases check that the word exists. \sy and \ay are alternatives for Synonyn and Antonym
9. Cross references such as \cf \mn \va should check that the entry exists and report on those missing.

When checking for cross references look in the citation form field as well as the lexeme form field so that cross references aren’t missed.

1. Watch out for \pc (Picture) fields that likely contain backslashes.
2. The program should be able to do a find and replace operation that uses regular expressions or plain text. The user should be able to restrict the scope of those changes to search only the data in certain markers, or to change only markers themselves.
3. The program should be able to report on the structure of the SFM data. That is to give information about the order of the markers in the records. A count of the number of occurrences of each structure. This should also report on the how that count changes if empty markers are ignored. One way this might work is that the software would look at the structure of the first entry and parse it into the sections that are Entry, Pronunciation, Sense, Example, Variant, and SubEntries. I’m wondering whether we can get the software to understand how FLEx will import the data, so that it can accept all those that parse unambiguously and show those that are unclear. Clean data should follow a limited set of structures when it takes into account the fact that some sections may be repeated.
4. Ideally the software would take the information it has been given about the Writing Systems, scripts and languages involved and at least check that an associated FLEx project seems to be correctly configured. Alternatively the software would read the WS and custom field settings and lists from FLEx to check that these are correct, that is they match the data, before importing.

**Functional requirements**

**Non-functional requirements**