# SoundFont Decoding Functional Specification

## Introduction

### Summary

The SoundFont decoding script will be a utility included with the TeensyAudio Wavetable codebase and will allow users to pull data for various instrumentation samples from SoundFont 2 files for use by the TeensyAudio Wavetable library.

### Requirements

See requirements documentation for detailed description of the functional requirements this document will implement.

### Existing System

Currently this functionality isn’t available to users of the Teensy library. The script to decode SF2 files will be a brand new addition to the TeensyAudio codebase.

### Terminology

**DAC** - Digital Analog Converter (the Teensy 3.2 is an example of a DAC).

**SF2** - SoundFont version 2 File Format (wavetable audio data wrapped w/ synthesis methods).

**Sample Array** - An array or list of individual byte samples to be produced by the SoundFont decoding script for use in the TeensyAudio Wavetable library.

**Attack** - Portion of the decoded sample array which plays immediately upon activation (e.g. piano key depression).

**Sustain** - Portion of the decoded sample array which can be looped indefinitely to produce a sustaining, consistent instrument tone.

**Release** - Portion of the decoded sample array which plays after the sustained section of a note, and will mark the natural end of a tone.

### References

Wavetable Synthesis on Microcontroller Requirements.docx - Rev 115

## Functional Description

**Use Cases**  
Most likely these will be kept in a separate document or CASE tool, referenced from the functional specification. Development of the use cases and functional specification should happen in parallel, where information from one feeds the other incrementally.

**Always avoid repetition.** The amount of detail in the rest of the functional specification will depend on the number of use cases that have been written.

Although important, use cases do not capture all functional requirements: this is why we need an encompassing functional specification. The availability of a separate document also discourages use case authors from putting too much detail in the use case (e.g. functional requirements instead of usage scenario text) or the wrong detail (e.g. boundary conditions), which are both common mistakes.

(Note this is a similar approach to the Unified Process “Supplementary Spec” which captures additional detail that should be kept separate from the use case).

Where the functional specification references a use case, always use the unique use case name (e.g. “Perform Order Entry”). Depending on the size of the system being modelled, you might also need to include the package name.

Similarly, if the use case references an item in the functional spec, always use the section and number of the functional item (e.g. “User Community, item 1.2.3.4”). If possible (given the constraints of the word processor or CASE tool being used) provide a hyperlink that takes the reader directly to the referenced item.

1. **Load Soundfont/SF2**
   1. Description: User provides something for the utility to process.
   2. Actors: User; Soundfont;
   3. Entry Conditions: N/A
   4. Exit Conditions: Relevant contents of the Soundfont are displayed on screen in some selectable format.
   5. Flow:
      1. User invokes *utility*, specifying local file path to desired soundfont/[SF2](#yth14kmarnhx).
      2. Necessary checking for compatibility and proper file structure is done by the utility. (Soundfont acceptance protocol is TBD)
      3. Success message displayed to user if compatible.
         1. Failure message displayed to user if Soundfont does not meet the protocol or in the event of a [File Read Error](#a93in3hj3lat).
         2. Exit.
      4. Continue to [Select Instrument](#x554sng9tf19).
2. **Select Instrument**
   1. Description: User needs to select an instrument(s) to use.
   2. Actors: User; Soundfont;
   3. Entry: Soundfont contents are displayed on screen.
   4. Exit: User has chosen *quit, or finish.*
   5. Flow:
      1. A prompt is displayed on screen to select an instrument, otherwise quit.
      2. User selects quit.
         1. Utility terminates.
      3. User inputs a selection.
         1. A prompt is displayed on screen to select an additional instrument, finish, or quit.
         2. If User selects an instrument
            1. go to 2.5.3.1
         3. If User selects finish
            1. Continue to Parse Samples.
         4. If User selects quit.
            1. Terminate utility.
3. **Parse Sample(s)**
   1. Description:
   2. Actors:
   3. Entry:
   4. Exit:
   5. Flow:



**User Community**  
The demographic of this project is anyone wishing to extract audio data from a SoundFont 2 file. This may include musicians, programmers, audio engineers, students, and casual “tinkerers”. The primary target system for this project is the Teensy 3.2 microcontroller and its related software, particularly for the purpose of wavetable synthesis.

**Error Handling**

***File read error:*** In the event that the program either cannot locate the target file, cannot properly extract audio data from the file, or finds the file to be corrupt or otherwise unusable, it will throw an exception without generating any output audio data.

***Invalid instrument:*** If the implementation of instrument selection (described below) admits the possibility of making an invalid selection, then the input should be ignored and the user should be prompted again.

**Help**  
Help will be accessible through the design/implementation documents and comments in code.

**Interface**  
The user will be able to execute the script from a command line, giving an SF2 file pathname as input. Assuming the file read is successful, the user will be prompted to select from a range of instruments encoded in the file. If the selection is valid, the script will produce (along with a confirmation message) an output .cpp file with 3 unsigned int arrays containing the byte string audio data extracted from the SF2 file for the given instrument. These arrays represent the 3 sections of the SF2 audio data (attack, sustain, release).

**Platforms**

This software will be supported on all OSes which support SF2 and Python 2.7.

**Portability**  
This utility should be usable by any version of Python > 2.7.

**Documentation**  
Developer documentation to be provided to list out how to utilize this utility, as well as the different functions/options that are exposed.