Module Interface Specification for ScoreGen

Team #7, Tune Goons
Emily Perica
Ian Algenio
Jackson Lippert
Mark Kogan

January 17, 2025

1 Revision History

Date	Version	Notes
2025-01-17	1.0	Initial version

2 Symbols, Abbreviations and Acronyms

- **MG**: Module Guide
- **M1**: Hardware-Hiding Module
- **M2**: User Interface Module
- **M3**: Score Generation Module
- **M4**: Raw Signal Processing
- **M5**: Audio Feature Extraction (note, key, sig, etc.)
- **M6**: File Format Conversions Module
- **M7**: Audio Recording and Playback Module
- **SRS**: System Requirements Specifications

Contents

3 Introduction

This document details the Module Interface Specifications for ScoreGen. A service designed to transcribe user-recorded musical compositions into accurate sheet music by determining pitch, duration, tempo, and more advanced musical features. The service also aims to provide a user interface to use and interact with the product. Complementary documents include the SRS and MG documents. The full documentation can be found on GitHub.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by ScoreGen.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
float	Note	a musical pitch, defined by a frequency in Hz
float	Duration	the length of a note, measured in beats or fractions thereof
float	Rest	a period of silence, defined by a duration
natural number	Tempo	the speed of the music, measured in beats per minute (BPM) as a natural number
float	Dynamic	the volume or intensity of a note as amplitude of a signal, e.g., piano (soft) or forte (loud)

The specification of ScoreGen uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, ScoreGen uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	-
Behaviour-Hiding Module	User Interface Module Score Generation Module File Format Conversion Module
Software Decision Module	Raw Signal Processing Module Audio Feature Extraction Module Audio Recording and Playback Module

Table 1: Module Hierarchy

6 MIS of Hardware-Hiding Module

6.1 Module

Hardware-Hiding Module

6.2 Uses

Provides abstraction for hardware (microphone and speakers), simplifying access for other modules.

6.3 Syntax

6.3.1 Exported Constants

- `DEFAULT_MICROPHONE`
- `DEFAULT_AUDIO_OUTPUT`

6.3.2 Exported Access Programs

Name	Input	Output	Exceptions
initializeMicrophone	None	None	InitializationError
initializeAudioOutput	None	None	InitializationError
readMicrophoneBuffer	None	rawAudioData	ReadError
sendToAudioOutput	audioData	None	PlaybackError

6.4 Semantics

6.4.1 State Variables

- `microphoneState`
- `audioOutputState`

6.4.2 Environment Variables

- `hardwareDriverLibrary`
- `deviceConfig`

6.4.3 Assumptions

- Functional hardware to take in user audio (i.e. microphone) and play out audio (i.e. speaker, headphones) is available.
- Necessary drivers are installed.

6.4.4 Access Routine Semantics

`initializeMicrophone()`:

- **Transition:** Sets `microphoneState` to "active".
- **Exception:** `InitializationError` if device fails to initialize.

`sendToAudioOutput(audioData)`:

- **Transition:** Sends `audioData` to hardware.
- **Exception:** `PlaybackError` if playback fails.

6.4.5 Local Functions

- `detectAvailableHardware()`
- `configureDeviceSettings(deviceType)`

7 MIS of User Interface Module

7.1 Module

User Interface Module

7.2 Uses

Defines the interaction mechanisms for uploading audio, recording, playback, and accessing generated PDFs.

7.3 Syntax

7.3.1 Exported Constants

- `DEFAULT_THEME`
- `MAX_UPLOAD_SIZE`

7.3.2 Exported Access Programs

Name	Input	Output	Exceptions
displayUploadInterface	None	None	RenderError
triggerPlayback	audioData	None	PlaybackError

7.4 Semantics

7.4.1 State Variables

- `currentScreen`
- `userPreferences`

7.4.2 Environment Variables

- `displayDriver`
- `inputDevices`

7.4.3 Assumptions

- User devices support modern UI rendering.

7.4.4 Access Routine Semantics

`displayUploadInterface()`:

- **Transition:** Sets `currentScreen` to "Upload".
- **Exception:** `RenderError` if rendering fails.

`triggerPlayback(audioData)`:

- **Transition:** Initiates playback of provided `audioData`.
- **Exception:** `PlaybackError` if audio fails to play.

7.4.5 Local Functions

- `validateUserInput()`
- `renderScreen(screenType)`

8 MIS of Score Generation Module

8.1 Module

Score Generation Module

8.2 Uses

Generates musical scores in compressed musicXML format (.mxl) from note sequences processed by previous modules.

8.3 Syntax

8.3.1 Exported Constants

- `DEFAULT_FONT_STYLE`
- `DEFAULT_PAGE_SIZE`

8.3.2 Exported Access Programs

Name	Input	Output	Exceptions
generateScore	noteSequence	.mxl file	GenerationError
customizeScoreSettings	settings	None	ValidationError

8.4 Semantics

8.4.1 State Variables

- `scoreSettings`

8.4.2 Environment Variables

- `fileSystemAccess`

8.4.3 Assumptions

- Input note sequences are formatted correctly.
- The file system is writable for saving .mxl files.

8.4.4 Access Routine Semantics

`generateScore(noteSequence)`:

- **Output:** An mxl file representing the musical score.
- **Exception:** `GenerationError` if input is invalid.

`customizeScoreSettings(settings)`:

- **Transition:** Updates `scoreSettings` with new values.
- **Exception:** `ValidationError` if settings are invalid.

8.4.5 Local Functions

- `validateSettings(settings)`
- `renderPDF(noteSequence, scoreSettings)`

9 MIS of Raw Signal Processing Module

9.1 Module

Raw Signal Processing Module

9.2 Uses

Processes raw audio signals to prepare them for feature extraction.

9.3 Syntax

9.3.1 Exported Constants

- `DEFAULT_SAMPLING_RATE`
- `DEFAULT_FILTER_SETTINGS`

9.3.2 Exported Access Programs

Name	Input	Output	Exceptions
filterSignal	rawAudioData	filteredAudioData	FilterError
adjustSamplingRate	audioData, rate	resampledAudioData	ResamplingError

9.4 Semantics

9.4.1 State Variables

- `currentSamplingRate`
- `filterParameters`

9.4.2 Environment Variables

- None

9.4.3 Assumptions

- Input audio data is in a readable format.

9.4.4 Access Routine Semantics

`filterSignal(rawAudioData)`:

- **Output:** Filters noise and returns cleaned audio data.
- **Exception:** `FilterError` if filtering fails.

`adjustSamplingRate(audioData, rate)`:

- **Output:** Resamples audioData to the desired rate.
- **Exception:** `ResamplingError` if resampling fails.

9.4.5 Local Functions

- `computeSpectralFeatures(audioData)`
- `applyFilter(rawAudioData, filterParameters)`
- `resample(audioData, rate)`

10 MIS of Audio Feature Extraction Module

10.1 Module

Audio Feature Extraction Module

10.2 Uses

Extracts meaningful features from processed audio for use in score generation.

10.3 Syntax

10.3.1 Exported Constants

- `DEFAULT_FEATURE_SET`
- `DEFAULT_WINDOW_SIZE`

10.3.2 Exported Access Programs

Name	Input	Output	Exceptions
extractFeatures	processedAudioData	featureSet	ExtractionError
configureFeatureSettings	settings	None	ValidationError

10.4 Semantics

10.4.1 State Variables

- `featureSettings`

10.4.2 Environment Variables

- None

10.4.3 Assumptions

- Input audio has been processed by the Raw Signal Processing Module.

10.4.4 Access Routine Semantics

`extractFeatures(processedAudioData)`:

- **Output:** Extracted features such as pitch, tempo, and dynamics.
- **Exception:** `ExtractionError` if feature extraction fails.

`configureFeatureSettings(settings)`:

- **Transition:** Updates `featureSettings` with new values.
- **Exception:** `ValidationError` if settings are invalid.

10.4.5 Local Functions

- `calculateTempoDynamics(audioData)`

11 MIS of File Format Conversions Module

11.1 Module

File Format Conversions Module

11.2 Uses

Handles conversion of input and output files between supported formats (.mxl ⇒ PDF).

11.3 Syntax

11.3.1 Exported Constants

- `SUPPORTED_IMPORT_FORMATS`
- `SUPPORTED_EXPORT_FORMATS`

11.3.2 Exported Access Programs

Name	Input	Output	Exceptions
importFile	filePath, format	mxl data	ImportError
exportFile	data, format, filePath	None	ExportError

11.4 Semantics

11.4.1 State Variables

- None

11.4.2 Environment Variables

- `fileSystemAccess`

11.4.3 Assumptions

- The specified file path exists for import operations.
- The export destination is writable.

11.4.4 Access Routine Semantics

`importFile(filePath, format)`:

- **Output:** .mxl data extracted from Raw audio input.
- **Exception:** `ImportError` if the file or format is invalid.

`exportFile(data, format, filePath)`:

- **Output:** Saves data in the specified format at the given file path.
- **Exception:** `ExportError` if writing fails.

11.4.5 Local Functions

- `convertToRawAudio(fileData, format)`
- `writeToFile(data, format, filePath)`

12 MIS of Audio Recording and Playback Module

12.1 Module

Audio Recording and Playback Module

12.2 Uses

Provides functionalities for recording audio through the microphone and playing back recorded or imported audio.

12.3 Syntax

12.3.1 Exported Constants

- `DEFAULT_AUDIO_FORMAT`
- `MAX_RECORDING_DURATION`

12.3.2 Exported Access Programs

Name	Input	Output	Exceptions
startRecording	None	None	RecordingError
stopRecording	None	rawAudioData	RecordingError
playAudio	audioData	None	PlaybackError

12.4 Semantics

12.4.1 State Variables

- `isRecording`
- `currentAudioBuffer`

12.4.2 Environment Variables

- `microphoneAccess`
- `speakerOutput`

12.4.3 Assumptions

- Microphone and speaker are functional and accessible.

12.4.4 Access Routine Semantics

`startRecording()`:

- **Transition:** Sets `isRecording` to true and starts capturing audio from the microphone.
- **Exception:** `RecordingError` if the microphone is unavailable.

`stopRecording()`:

- **Output:** Captured audio as raw data.
- **Transition:** Sets `isRecording` to false.
- **Exception:** `RecordingError` if no recording is in progress.

`playAudio(audioData)`:

- **Output:** Plays the specified audio data through the speaker.
- **Exception:** `PlaybackError` if playback fails.

12.4.5 Local Functions

- `captureMicrophoneInput()`
- `sendToSpeaker(audioData)`

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

Appendix — Reflection

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   Emily: Since a lot of the code has already been written (or at least planned out), the process of identifying modules and their behaviours for this deliverable went quickly and made the whole process a lot simpler.

   Mark: This document was helpful in creating a better understanding of the future of ScoreGen at a more practical, rather than conceptual level.

   Ian: Splitting the work and dividing into sub teams helped speed up the process of breaking down the two documents and their sections. This was good for work efficiency. Communication before the deadline of this deliverable was productive and informative.

   Jackson: I think this deliverable was a good step to lay out all of our plans for our code. This will honestly benefit us in our development of the app itself, as having this to look back at will benefit us greatly.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   Emily: Making the traceability matrix between modules and anticipated changes proved to be a bit of a challenge. We were directed to identify anticipated changes that ideally affect only a single module, which forced me to think more critically about each AC and why they are needed.

   Mark: As some modules are in the process of completion it was difficult to define specific terms that hadn't yet been implemented, thus completing the documentation

required a lot of communication and assumptions. It is also highly likely that changes will need to be made in future revisions of this document to align with design choices that occur later during implementation.

Ian: One pain point was deciding how abstract some of the descriptions of the module secrets/services in the module guide. A balance had to be struck between being able to adequately describe something vs explaining too many details/choices. This was easily solved by re-reading the MG template, and by taking a look at previous students' work and how they handle this.

Jackson: This deliverable was due very soon after the winter break, and getting back into the flow of school plus reconnecting with the group and getting everyone on the same page was tough. Additionally, breaking down the modules into their specifics proved tough, as creating detailed information about our implementation at this stage. At the same time, the app is still being developed, which was difficult to do.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

   The majority of design decisions made up until this point did not stem from speaking to clients. The main decisions made such as the algorithmic choices and which file formats to support were decisions made based on the team's knowledge of signals and systems and human-computer interfacing. These decisions included selecting the Fast Fourier Transform for pitch detection due to its performance and efficiency and supporting widely used file formats for distribution purposes (PDF) and musical representation (musicXML).

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), if any, needed to be changed, and why?

   N/A

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

   Our project aims to create as strong an ability as possible for non-technical musicians to create highly detailed notation, but there are intricacies that are extremely difficult to extract from audio alone. Features like staccato, crescendo, chords, grace notes,

or tempo changes are difficult to differentiate from variance that occurs from regular human playing. Tackling this issue effectively would probably best be done with very advanced signal processing and personally trained, or fine-tuned machine learning models. Given more time, it would also be helpful to implement advanced options for users to maximise precision. If for example there is a music piece with lower confidence sections, such an area where there is a similar likelihood of a note being a fast-played 16th note, or a grace note, it could be possible for the user to toggle through most likely interpretations with playback to determine the ideal representation of their playing.

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

   We considered several design solutions, including purely rule-based algorithms and advanced signal processing techniques. The rule-based approach would have been easier to implement but lacks the flexibility to interpret complex musical nuances like dynamics and articulation. Advanced signal processing, while more capable of handling these intricacies, would require more computational resources and expertise in the field. After weighing the tradeoffs, we chose a hybrid approach that combines signal processing with rule-based methods. This design provides a balance between accuracy, flexibility, and resource efficiency, ensuring a strong foundation for transcription while leaving room for future refinement.