## Project Overview

The Music Maker team will create an app, in C#, where a user can click notes on a virtual piano and have those notes appear on a score which can then be played back. The score can be printed and/or saved as MusicXML, which allows compatibility with most other modern notation programs.

## Project Responsibilities

Assigned project responsibilities are as follows:

Leith McCombs: Project management, data architecture, save and load, class hierarchy and interface design.

Nicholas Grayson: Playback and printing.

Jacob St Pierre: GUI, including the piano display, score window, and pallets/toolbars/dialogs for setting note values, time signatures, etc.

## System Requirements

* Windows 7 or higher
* .NET Framework 4.0 or higher
* Sound card and speakers

\* These requirements are a temporary estimate and subject to change.

## Dependencies

The following third-party frameworks, controls, and libraries are (or may be) required for this project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Description** | **Used for** | **Available from** |
| MusicXML | An XML schema that describes musical information | File format, internal data model | <http://www.musicxml.com/for-developers/> |
| PSAM Control Library | A library of XAML controls for displaying musical information from MusicXML | Score display | <http://www.codeproject.com/Articles/89582/PSAM-WPF-Control-Library> |
| MIDI | The Musical Instrument Digital Interface standard | Format held in memory for playback, may also be file format option for save | N/A |
| C# MIDI Toolkit | A toolkit for creating MIDI applications in C# | \*May\* use for converting MusicXML to MIDI for playback | <http://www.codeproject.com/Articles/6228/C-MIDI-Toolkit> |
| MIDI.NET | Another toolkit for creating MIDI applications in C# | \*May\* use for converting MusicXML to MIDI for playback | <http://midinet.codeplex.com/wikipage?title=Samples&referringTitle=Documentation> |
|  |  |  |  |

## Scenario Requirements

The scope of supported scenarios will depend on how quickly the core functionality can be built. Therefore, we will group the scenarios by priority. This does not necessarily mean that they will be completed in priority order, but rather that the importance of a scenario will be weighed against the amount of time necessary to support it.

### Pri 0

The product cannot release without completing these.

1. User inputs notes, sees notes displayed on a staff.
2. User saves file, closes it, and re-opens it later.

### Pri 1

All of these should be fully supported if at all possible.

1. User composes song, saves it, and prints to paper.
2. User listens to playback of partial or completed song, hears an error, and changes the notes on the staff. *I Think the change notes on staff needs to move to priority 2. Going to take more work than to get working correctly.*
3. Song includes chords or multiple lines of music, and standard musical artifacts such as rests, ties, and fermatas.
4. Song includes changes in time signature and key signature.

### Pri 2

We will support as many of these as we can.

1. User enters notes in real time.
2. Song includes lyrics and chord symbols.
3. User saves file and opens in another notation program, or opens an existing file from another program in MusicMaker.
4. User decides to save an existing piece in a different key.

### Pri 3

These are the scenarios that we would like to support if we can finish the base functionality, because they will add unique value.

1. Singer or non-chordal instrumentalist writes a melody, uses programmatic tools to come up with the chords.
2. Choral or orchestral composer writes a polyphonic piece and uses programmatic tools to check it against standard music theory rules.
3. Guitarist writes a melody and cord symbols, software converts the chords into four-part vocal harmony, guitarist leaves the strong beats mostly in place and tweaks the weak beats to make the individual lines flow.
4. User plugs in a MIDI instrument and uses it for note input.

## Feature Requirements and Implementation

Because the scenarios are worded to allow for flexibility, there are multiple feature and implementation choices that could meet those requirements. Given the complexity inherent in the project, much of the functionality will depend on third party components, which are still being explored, so this section is subject to change.

### Note input (pri 0)

Ideally, notes may be entered on a virtual keyboard or by mappings from the computer keyboard. Supporting both options is pri 1. Note values (quarter, half, etc.) may be specified by an on-screen palette or by keyboard mappings. Again, supporting both options is pri 1.

Implementation: Virtual keyboard is a series of XAML rectangles that map to an event handler that specifies the note. Note value is a static variable, where the most recently set value is assigned to the current note. This data is then saved into an XmlDocument object with the MusicXML schema.

### Display on staff (pri 0)

Implementation: Current plan is to use an IncipitViewer control from the PSAM Control Library. It automatically displays MusicXML data on a staff.

### Load and save (pri 0)

Implementation: Files will be saves as MusicXML documents with a .mml extension.

### Print (pri 1)

This will probably be done with the standard .NET print calls, but has not yet been fully investigated. The contents of the score window will have to be formatted for printing first.

### Playback (pri 1)

In its simplest form, the app will save the current document and play it as a MIDI file. Being able to start playback in the middle of a song is pri 2. Also, each note will be played on entry (pri 1).

Implementation: This is probably the most difficult feature to implement. MusicXML is designed to be mappable to MIDI, but so far we have not found any pluggable player controls to do this for us. There are several libraries to work with MIDI in C#, so we will probably work with those if we can’t find something simpler.

### On-staff editing (pri 1)

A user should be able to select a note on the staff and change its pitch and duration. This may be done through a context menu, dialog box, or through drag and drop, depending on what the PSAM Control Library supports. Further investigation is needed.

In the event of notes being added, deleted, or changes in duration, the notes afterward will have to be adjusted accordingly. This may involve truncating notes or adding rests, or allowing the remainder of the line to be pushed along, or providing an option for the user to decide which behavior to use.

### Multiple lines of music (pri 1)

The user should be able to create chords (multiple notes at the same time on a single staff) and have multiple concurrent staves. The exact UI for this is TBD.

Implementation: Both chords and concurrent lines are fully supported by MusicXML, MIDI, and the PSAM Control Library.

### Rests and artifacts (pri 1)

The user should be able to create rests and ties (pri 1), fermatas, and other standard music notation artifacts (pri 2). The exact UI for this is TBD.

Implementation: These are fully supported by MusicXML, MIDI, and the PSAM Control Library.

### Time and key signatures (pri 1)

User should be able to set time signature and key signature both at the beginning (pri 1) and middle (pri 2) of the piece. The exact UI for this is TBD.

Implementation: I believe these are fully supported by MusicXML, MIDI, and the PSAM Control Library, but will have to verify.

### Realtime note entry (pri 2)

User sets the tempo, time signature, and note value threshold, then clicks an on-screen button and a metronome sound starts. They then enter notes in real time by keyboard mappings (pri 2) or with a MIDI instrument (pri 3). The note durations are quantized into the nearest 16th note, or whatever minimum value threshold is set by the user.

Implementation: The keyboard mapped version will necessarily be limited in pitch range by the available number of keys available for mapping. Most likely, a default octave would be supplied and notes would be assumed to be the nearest octave to the last note entered unless the user hit the up or down arrow key prior to entering the note. Note values would be determined mathematically by comparing key press times to metronome clicks and subdividing accordingly.

MIDI instrument implementation is probably out of scope unless we find a plugin to simplify the process.

### Lyrics and Chord Symbols (pri 2 or lower)

This would be a high priority for a shipping product but is less important for a class project, so it will only be included if the functionality is built into the PSAM Control Library. The exact UI will depend on what is provided by the library.

### File portability (pri 2)

By using MusicXML as the file format, files are automatically compatible with most modern notation programs.

### Transposition (pri 2)

User should be able to create or open a piece and then change the key with a single interaction. Standard transposition (moving notes up or down the staff but staying in the same mode) is pri 2, transposition in place (keeping the same staff positions but changing key signature) is pri 3. Exact UI TBD.

Implementation: This would all be done by manipulating the XmlDocument object and then refreshing the score UI.

### Harmonization tools (Pri 3)

Since these are likely to be out of scope, we will not spend a lot of time defining them right now. The general UI concept should be similar the that of a spell-checker in a word processing program. The user would have a dialog box to select which rules to use or what chord set to work from and then the program would apply the rules. It is possible that one or more of these features will be prioritized over some of the pri 2 features because they are more interesting from a project standpoint, though the pri 2 features would be more important to users.

### MIDI instruments

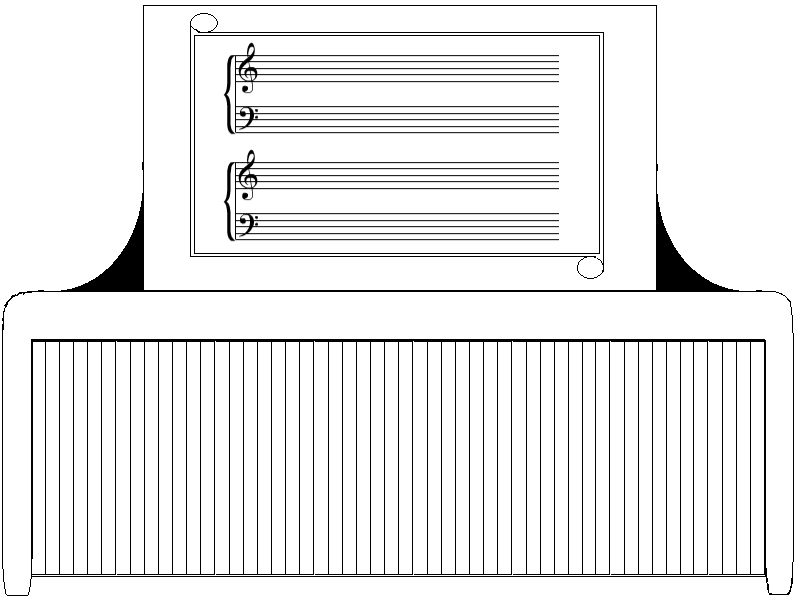
Compatibility with MIDI instruments would make the program much more appealing to serious musicians but is probably too complex to attempt in an 8-week class unless we can find a pluggable control to handle the interface.

# UI Specification

The MusicMaker program is a Windows application with UI components created in XAML. The GUI interface will be created in WPF and include event handlers on the cs script for the main window.

## Main Window

The Main window will revolve around a piano setup. The piano will have a full size keyboard of white and black keys. Piano keys will change color when moused over. Keys will resume original color when mouse leaves the vicinity of the key. Keys will change to a third color when clicked and will go back to the mouse over color when unclicked. The secondary window located above the piano keyboard will be a music sheet. The music sheet will include multiple lines of music.



## Palettes and Toolbars

Sheet music will be split in two parts. One will be bass and one will be treble. Program will detect correct keys and placed in correct area..

Toolbar is outside the scope of pri 0, 1, 2 making it a pri 3. Time permitting the tool bar will have a note for the ability to drop and place note to sheet music.

## Menus

Program will have multiple menu buttons on screen. These will be located directly below the sheet music and above the key board. They will include an open, close, save, play, pause, stop at the minimum. Time permitting additional menu buttons will be placed like rewind and forward.

## Dialog boxes

Program will have user input through drop down menu for music time. User will be able to select time the music time they wish to use. This will be used by the program for the speed of playback. Note duration will fit in program in some aspect. How has not been determined.

# Milestones

The following milestones apply to the MusicMaker project. These may be adjusted as the project progresses.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Milestone | Date | Jacob | Nicholas | Leith |
| M0 | June 3 | Virtual keyboard laid with keys named to notes and click mapped to a shared event handler stub, score window in place. | Final decision made on how to generate MIDI data, Xml2Midi function begun. | Notes and durations input into XML document, document wired to score display. |
| M1 | June 17 | All GUI objects for pri 0 and pri 1 features in place | Basic playback mostly working. | All inputs integrated into XML document, save and load complete, basic utility functions in place to access music data as needed. |
| M2 | July 1 | UI elements visually refined, overall look and feel complete, additional UI objects added as needed, work on P2 and P3 features if time available. | Scores printing correctly, playback complete, work on P2 and P3 features if time available. | On-staff editing complete, work on P2 and P3 features if time available. |
| Complete | July 8 | Testing, bug fixes, and documentation. | Testing, bug fixes, and documentation. | Testing, bug fixes, and documentation. |