mTunes: A Midi Music Development Language

HeeWook Lee Sharon Price[1] Hideki Sano Huitao Sheng
{hl450, sbp2001, hs2160, hs734}@columbia.edu

[1] Group Leader

1.1 Introduction

The mTunes music development language is designed to alleviate the void for simple and inexpensive midi composition and alteration software. Most home users, as well as many workstations (example A – Columbia's PSL Lab), have music playing software. Users can play mp3, midi, wav, aiff, and other popular audio format files. However, none of the freely available software allows these users to compose their own music or edit pre-existing songs to create their own masterpieces. For a small fee ranging from $100 to $5000, professional quality audio editing software can be purchased. With the professional level of tools comes the professional level of unusability. However, this is unrealistic for most enthusiasts who merely want to edit existing files or create short songs of their own. And so mTunes was spawned.

1.2 mTunes Description

mTunes: An easy-to-use, object-oriented, portable, reusable, midi composition and transformation language. Allows users to add and remove instruments, entire tracks, and single notes, as well as perform transformations on the entire file, such as changing the scale.

1.2.1 Easy-To-Use

Music composers already have to deal with hundreds of musical commands. mTunes has a small, simple command set that is easy to master. Instead of having composers who are not computer-savvy trying to use foreign interfaces, they only need to remember a few easy keywords and their associated parameters. This allows composers to focus on their music instead of on overly complicated software.

1.2.2 Object-Oriented

mTunes composers create new instances of the types of instrument they want to use or add to an existing song. To have the instruments play, the play method is called on the instruments and given a few easy parameters. Up to sixteen (16) instrument objects can be placed inside a track, and then operations can be performed on the entire track. Similarly, existing files are broken down into tracks to be edited. Notes and other events can be extracted from the different tracks, regardless of the source of the tracks.

1.2.3 Portable

The interpreter is written in Java, so it can be used on almost every operating system and architecture still in use today. This provides mTunes with seamless portability. The mTunes code itself is written in a text file, which enables it to be transferred to any other computer.

1.2.4 Reusable

Even with an easy to use language, reusability of code is important; users can always spend extremely long periods of time coding to produce intricate programs. With mTunes, previously written code, with little or no changes, can be used to create new songs or edit other pre-existing songs.

1.3 Language Features

1.3.1 Data Types

Each new midi file being created is called a composition. For simplicity, there are only three data types available for users of mTunes – *instruments*, *files*, and *locations*. The *file* data type is an existing midi file that the user includes in the composition for either background music or for modification. *Locations* consist of the measure and beat where the note starts playing.

1.3.2 Basic Commands

The most important three commands are: play; start; and stop. The play command simply plays one note for the indicated length. To begin and end playing a file, the start and stop commands are used. Many of the most common transforms performed on music are also enabled, such as declaring the speed in the form of beats per minute (bpm) and transcribing an entire song.

1.3.3 Flow Control

Almost every song contains some element that is repeated, which inspired the repeat command. Any code inside the repeat command is played consecutively the number of times specified. Similarly, the pattern command allows users to define and name a specific pattern of notes and operations, with or without specified instruments. The pattern is then played the same way a regular note is. If there are commands for which no instrument is specified, the pattern must be played with an instrument parameter. The inputted instrument is then used in all the operations for which no instrument was specified.

1.3.4 Saving and Playing Compositions

Should the user start mTunes with a .midi file instead of an mTunes (.mT) file, mTunes automatically just plays the song without attempting to compile, &c. Also, at any time during or after the playback of a newly compiled composition, the user can save the composition as a regular midi file (.midi).

1.4 Example Syntax

1.4.1 Sample comment

Note: Everything after the ; is ignored in a line

;<comment>

```
;this is a comment! YAY!
```

1.4.2 Creation of an instrument and a file

<type> <filename/instrument type> <name>;

```
instrument guitar fender1;
file Beethoven's9th.midi b9th;
```

1.4.3 Playing a note and a file

play <name> <note>|<length> <location>;

```
play fender1 C#|half 4|1;
start b9th 4|1;
stop b9th 14|1;
```

-or-

location <name> <measure>|<beat>

```
location location1 4|1;
play fender1 C#|half location1;
start b9th location1;
stop b9th 14|1;
```

1.4.4 Playing a note using repeat option

repeat <name> <note>|<length> <start location> <# times to repeat>;

```
repeat fender1 C#|half location1 5;
```

1.4.5 Creating a pattern

Note: Locations inside patterns are relative to start of pattern

pattern <name> { <commands inside pattern> }

```
pattern GeneralPattern {
       play fender1 C#|half location1;
       repeat B|half location1 5;
}
```

1.4.6 Playing a pattern

play <pattern name>(<instrument, if required>) <location>;

```
play GeneralPattern(fender1) location1;
```