Songlib: control functions

John C. Lusth

Revision Date: April 20, 2017

Controlling output

There are a number of functions for controlling the output of played notes.

setTime

```
void setTime(int beatsPerMeasure,int noteValue);
```

Sets the time signature of a song. The *beatsPerMeasure* argument is used by the keepingTime facility.

setPrimaryEmphasis, getPrimaryEmpahasis

```
double setPrimaryEmphasis(double amplification);
double getPrimaryEmphasis(void);
double setSecondaryEmphasis(double amplification);
double getSecondaryEmphasis(void);
```

Songlib can place emphasis on the first beat of a measure. It does so by increasing the amplitude of notes played at this location. To enable this feature, send an amplification value other than 1. **Songlib** can also place an emphasis on the middle beat of a measure (assuming there is a middle beat) using *setSecondaryEmphasis*. Usually, the primary emphasis is set to be larger than the secondary emphasis. Sending amplification values less than one to either function causes the emphasized beat to be played softer than the surrounding beats.

Both *setPrimaryEmphasis* and *setSecondaryEmphasis* return the previous amplification values.

setSustain, getSustain

```
double setSustain(double fade);
double getSustain(void);
```

Sustain governs how long a note continues to play after it has played for the requested number of beats. It is often undesirable for a note to abrubtly end as this often results in audible clicks upon playback. A better strategy is to have the note fade out. A sustain value near very close to one results in a long fade-out while a sustain value further away from one results in a quicker fade-out.

Setting a sustain value of 1.0 means 'no fade' and previous notes will continue playing with no diminishing of amplitude until the note data is exhausted. A value of 0.0 means note stops playing abruptly. Good values usually range between 0.999 and 0.99995. The default value is 0.999. A value of 0.99995 gives an effect similar to holding the sustain pedal down on a piano.

The return value of *setSustain* is the previous sustain value.

setStride, getStride, setStrideSlop

```
double setStride(double delay);
double getStride(void);
```

Sets the delay between the starts of the individual notes of a chord to *delay* beats. A delay value of 0.0 means all notes of a chord will start at the same time. A positive delay means the first note in the chord starts immediately, the next note in the chord starts *delay* beats later, the next note *delay * 2* beats later still, and so on. The use of a positive stride gives a more realistic chording. The default stride delay is 0.25 beats.

The return value of *setStride* is the previous value of *delay*.

The function *setStrideSlop* can be called to randomly vary the delay between chord notes. The single argument to *setStrideSlop* is a real number that represents seconds. Given a value of s, the function will adjust the stride value between notes by a random value chosen between $-s$ and s.

See chord for more information about chords.

setSkipBeats, getSkipBeats, setSkipSeconds, getSkipSeconds

```
double setSkipBeats(double beats);
double getSkipBeats(void);
double setSkipSeconds(double seconds);
double getSkipSeconds(void);
```

Causes the first *beats* of the entire output to be thrown away. Useful for when you are working on the interior of a track and you don't want to listen from the very beginning to assess your changes.

The return value of *setSkipBeats* is the previous value of the skip.

The alternate forms, *setSkipSeconds* and *getSkipSeconds*, are like *setSkipBeats* and *getSkipBeats*, only the units are in seconds.

setAmplitude, getAmplitude

```
double setAmplitude(double level)
double getAmplitude(void);
```

Sets the overall amplitude of output to that of *level*. Returns the previous amplitude. The default level is 1.0 meaning the notes are output as read. To increase the volume of the output, choose a level greater than 1.0. To decrease the volume, choose a non-negative level less than 1.0.

The function *setAmplitude* returns the previous amplitude setting.

setTempo, getTempo

```
int setTempo(int beatsPerMinute);
int getTempo(void);
```

Sets the tempo of a song. The default tempo is 132 beats per minute. Setting the tempo with a value higher than 132 speeds up the song while setting the tempo to a lower value slows down the song.

The return value of *setTempo* is the previous tempo setting.

setKey, getKey

```
int setKey(int key);
```

```
int getKey(void);
```

Sets the key signature of a song. The argument *key* is one of the predefined variables: **C**, **Cs**, **Db**, **D**, **Ds**, **Eb**, **E**, **F**, etc.

See also: keySignatures .

setDrawRamp, getDrawRamp

```
double setDrawRamp(int double);
double getDrawRamp(void);
```

Sets the time of transition between successive notes in a draw.

See also: draw .

beginCrescendo, endCrescendo

```
void beginCrescendo(double ramp);
void endCrescendo(void);
```

These functions are always used in pairs; in between the calls, the songlib output steadily increases or decreases in volume depending on the value of *ramp*. Values greater than one cause an increase in volume; *ramp* values less than one cause a decrease in volume. A ramp value of 2 eventually doubles the amplitude values over the span; a ramp value of 0.5 eventually halves the amplitude values over the span. Note: due to the non-linear nature of perceived volume, doubling (halving) amplitude values does not double (halve) the perceived volume.

beginRitardando, endRitardando

```
void beginRitardando(double ramp);
void endRitardando(void);
```

These functions are always used in pairs; in between the calls, the songlib output steadily increases or decreases in tempo depending on the value of *ramp*. Positive values cause a decrease in tempo; negative *ramp* values cause an increase in volume. When a note with length *b* beats is played, the tempo is reduced by *b* * *ramp* after the note concludes. When a chord is played, the tempo is decreased after the last note of the chord is played. The behavior of this function is undefined if a call to *backwards* is made in between the *beginRitardando* and *endRitardando* calls.

setInterp, getInterp

```
int setInterp(int type);
int getInterp(void);
```

The three interpolation methods available in songlib are linear (type 1), sinusoidal (type 2), and cubic (type 3). The smaller the type number, the faster and more crude the interpolation method. The *setInterp* function returns the previous interpolation type while the *getInterp* method returns the current type. The default type is sinusoidal.