Songlib: drums

written by: Song Li Buser

Revision Date: April 18, 2017

Drum support

Songlib provides three approaches beyond the play/splay/drum set of functions. They are *drumline*, whereby you play a set of predefined patterns, *drumkit*, whereby you provide separate patterns that are played by each instrument in the drumkit, and *processtab*, whereby you specify a grand pattern describing all instruments that is subsequently converted into **songlib** code. The *drumline* approach is the simplest but least versatile, while the *processtab* approach allows more creativity but requires no programming. The *drumkit* approach lies allows creativity, but requires programming.

Drumline

The *drumline* approach utilizes the *drumline* function, which plays predefined patterns supplied in the function call. The function utilizes a single sample pack which holds single hits on various percussive instruments, according to the drumset standard (see noteNotations.html).

Here is an example call to *drumline*:

```
drumset = readScale("/usr/local/samples/share/drumset/","hera_");
drumline(4,drumset,d44c,0);
```

The first argument to *drumline* is the number of measures you wish the drumset instruments to be played. In the example, the instruments are to be played for 4 measures. The second is the instrument scale do be used. The third argument is the pattern to be used. Currently there are nine patterns:

- d44a
- d44b
- d44c
- d44d
- d44e
- d44f
- d44g
- d44h
- d44i

All patterns starting with 'd44' will each span one measure in 4:4 time. Eventually, there will be d34, and d68 patterns as well.

The last argument to *drumline* is the pattern to play for the last measure, if you wish to have a different pattern for the last measure. If a zero is passed the third argument is used for the last measure (*i.e.*, all measures are the same pattern).

If you wish, you can write your own patterns to send to *drumline*. The patterns you send to *drumline* should look something like:

```
void
pattern(int drumset)
    {
    ...
    }
```

For example patterns, see:

```
~/songlib/lib/drumlines.c
```

The following pattern demonstrates how to have the pattern change from one measure to the next:

```
void
d44a(int drumset)
    {
    int totalMeasures = 2;
    static int mode = 0;

    if (drumset < 0)
        {
        mode = 0;
        return;
        }

    switch (mode)
        {
        case 0:
        drum(I,drumset,TOM_MIDDLE);
        drum(Qd,drumset,TOM_MIDDLE);
        drum(H,drumset,TOM_LOW);
        break;
        case 1:
        drum(H,drumset,TOM_LOW);
        drum(I,drumset,TOM_LOW);
        drum(Qd,drumset,TOM_MIDDLE);
        break;
        //add more measures/cases here
        }

    mode = (mode + 1) % totalMeasures;
    }
```

Note that sending a nonsensical drumset (e.g. -1) resets the pattern back to the beginning. Otherwise, subsequent calls cycle through the total number of measures

Drumkit

The *drumkit* approach utilizes the following set of function:

- *drumkitCrash*
- *drumkitHHOpen*
- *drumkitHHClosed*
- *drumkitHHPedal*
- *drumkitRide*
- *drumkitSnare*
- *drumkitTom*
- *drumkitTomHi*
- *drumkitTomLo*
- *drumkitKick*
- *drumkitRim*
- *drumkitStick*
- *drumkitCowbell*

An example call to *drumkitHHPedal* might look like:

```
drumkitHHPedal(10,0.75,"-x-x-x-x-x-x-x-x","xxxx",(char *) 0);
```

The first argument is the number of repeats. In this case, the supplied patterns will be repeated 10 times. The second argument is the amplitude. In the example, the amplitude is being set to three-quarters value. The next arguments are the patterns to be played by the instrument; patterns are terminated with a zero. Each pattern corresponds to a measure and the length of the pattern specifies the number of beats in the measure. For example, the pattern:

```
"-x-x-x-x-x-x-x-x"
```

says that the instrument should be played using sixteenth notes. A `'x'` in the pattern means the instrument is to be played, while a `'-'` in the pattern indicates a rest. The pattern

```
"x--x"
```

means the instrument should be played using quarter notes with pattern "*hit rest rest hit*". All the other drumkit functions work in a similar fashion.

Note, the entire pattern set is played before the set is repeated.

Here is an example of a 10 measure rock-n-roll beat:

```
drumkitStick(1,1,"xxxx",(char *) 0); //intro
spot = getLocation();
drumkitCowbell(10,1,"g-g-g-g-g-g-g-g-",(char *) 0); setLocation(spot);
drumkitKick   (10,1,"x---x---x---x---",(char *) 0); setLocation(spot);
drumkitSnare  (10,1,"----x-------x---",(char *) 0); setLocation(spot);
drumkitHHPedal(10,1,"--x---x---x---x-",(char *) 0); setLocation(spot);
drumkitRim    (10,1,"---x-----x-----x",(char *) 0); setLocation(spot);
drumkitTomHi  (10,1,"x---x-----x---x-",(char *) 0); setLocation(spot);
drumkitTom    (10,1,"-------x--------",(char *) 0); setLocation(spot);
drumkitTomLo  (10,1,"-x------x--x----",(char *) 0);
```

In a pattern, only certain characters are significant; any other character is considered a rest. Thus, the following two patterns are equivalent:

```
"x---x-----x---x-"
"x234x23412x412x4"
```

You may prefer the latter formatting for patterns.

The default drumkit should have been installed when you installed **songlib**. If you wish to use different instruments than those in the default drumkit, you can get and set each instrument individually with the following functions:

- *getCrash* and *setCrash*
- *getHHOpen* and *setHHOpen*
- *getHHClosed* and *setHHClosed*
- *getHHPedal* and *setHHPedal*
- *getSnare* and *setSnare*
- *getTomHi* and *setTomHi*
- *getTom* and *setTom*
- *getTomLo* and *setTomLo*
- *getKick* and *setKick*
- *getRim* and *setRim*
- *getStick* and *setStick*
- *getCowbell* and *setCowbell*

All of these functions behave similarly; here is an example call to *setCrash*:

```
inst = readScale("/usr/local/share/samples/mydrumkit/","crash_");
oldCrash = setCrash(inst);
```

Note, the *drumkit* function assumes that there are exactly 11 samples in each instrument sample pack, located at notes 0, 12, 24, 36, 48, 60, 72, 84, 96, 108, and 120. Things will go badly if this is not the case for your custom drumkit.

Other drumkit functions

For your convenience, there is a generic form of the drumkit functions:

```
drumPat(instrument,amplitude,repeats,pattern1,...,patternN,(char *) 0);
```

With *drumPat*, the instrument is passed in as the first argument. The *roll* function can be used to generate a drum roll:

```
roll(duration,instrument);
```

The speed of the roll is set with the *setRollSpeed* function; the default speed is a 32^{nd} note, meaning each hit in a roll lasts $\frac{1}{32}$ of a beat. A periodic emphasis can be added with the *setRollEmphasis* function; the default emphasis is 4, meaning every fourth hit is a little louder than the others.

Three more functions in the drumkit family are:

```
ghost(duration,instrument);
flam(duration,instrument,emphasis);
drag(duration,instrument,emphasis);
```

The *ghost* function plays a very quiet hit. The *flam* function adds a softer grace note before the main hit. The *emphasis* argument dictates how much of an accent is given to the main hit. A value greater than one makes the main note louder, while a value less than one makes the main note softer. The *drag* function is similar to *flam*, except two grace notes precede the main hit.

As with all other drumkit functions, the instrument passed to *drumPat*, *roll*, and *grace*, would have an eleven note sample set with the notes occurring at multiples of 12.

Pattern characters

As indicated above, an 'x' in the pattern indicates a hit and a '-' indicates a rest. Here is a list of all the special characters that can be placed in a pattern:

`x`	a regular hit
`X`	an emphasized hit
`g`	a ghost note
`G`	a ghostier note
`r`	a roll
`R`	an emphasized roll
`f`	a flam (one ghost note)
`F`	an emphasized flam
`d`	a drag (two ghost notes)
`D`	an emphasized drag
`^`	regular hit, one semitone higher
`A`	emphasized hit, one semitone higher
`v`	regular hit, one semitone lower
`V`	emphasized hit, one semitone lower

As stated earlier, any character not appearing in the above table is treated as a rest.

Processtab

One can generate **songlib** programs using a text description similar to the patterns used by the *drumkit* approach. See tab for more information.