Ongaq JS: An elementary library for programming music

Hiroyuki Takakura
CodeNinth Ltd, Tokyo
708 9-20 Higashi-Kishicho, Urawa, Saitama
takakura@codeninth.com

ABSTRACT
In this paper, I describe concepts and usages of Ongaq JS, a new elementary JavaScript library for programming music. This library makes it possible for many people, especially beginners or young students, to program music using various sound resources.

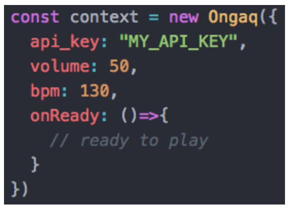
We, CodeNinth Ltd., are going to make this library accessible as OSS [1] and open a website [2] to read documents, view samples and manage licenses in 2019.

1. CONCEPTS
The main concept of Ongaq JS is to make it easier and more pleasant to program music. These days, many people want to acquire skills of programming and create what can keep motivating them to study. Therefore, we have developed Ongaq JS for this purpose.

2. USAGES
In this section, I describe usages of Ongaq JS in order of appearance in general program.

2.1 Create An Account
Before starting to write code, we have to create an account of Ongaq JS and obtain an API key. This API key is required for fetching sound resources from our server. The details of the sound resources are described in my previous papers [3].

2.2 Prepare A Context
Prepare a context of Ongaq JS by initializing an Ongaq object (See Figure 1). This object imports Part objects which constitute music compositions. We can play and pause them by methods on the Ongaq object.

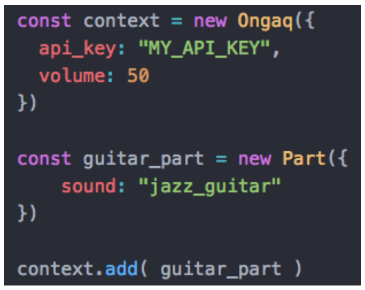


```
const context = new Ongaq({
  api_key: "MY_API_KEY",
  volume: 50,
  bpm: 130,
  onReady: ()=>{
    // ready to play
  }
})
```

Figure 1. Initializing an Ongaq JS context

2.3 Add Part Objects
Create Part objects to form our music compositions. A Part object can be compared to a member of a band. We assign each Part object a sound resource and attach them to the Ongaq object. (See Figure 2)

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

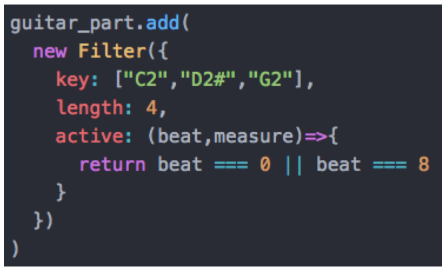


```
const context = new Ongaq({
  api_key: "MY_API_KEY",
  volume: 50
})

const guitar_part = new Part({
    sound: "jazz_guitar"
})

context.add( guitar_part )
```

Figure 2. Attaching a Part object to the context

2.4 Define Behaviors with *Filter* Objects
We can add Filter objects to the Part objects to define their behaviors. There are various types of Filter objects which define different behaviors. For example, a “note” type Filter defines when and which musical scales are played and a “pan” type Filter defines when and from which direction sounds come. Note that for some instruments, names like “hihat” or “kick” may be used instead.

In Figure 3, the Part object is defined to play “C2”, “D2#”, “G2” at beat 0 and beat 8 for 4 beat length. The function which is assigned to “active” property is called right before each beat while the context is playing, receiving the beat index as argument.



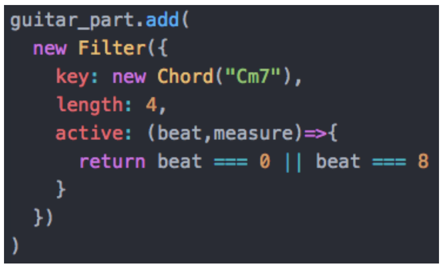
```
guitar_part.add(
  new Filter({
    key: ["C2","D2#","G2"],
    length: 4,
    active: (beat,measure)=>{
      return beat === 0 || beat === 8
    }
  })
)
```

Figure 3. Adding a Filter object to the Part object

By default there are 16 beats in a measure. This can be overwritten by settings of Part objects.

2.5 Chord: Helper Object
Chord is a helper object which provides a list of musical scales consisting of a chord. It also has some methods to shift or rearrange them. This makes it easier for developers to obtain harmony.

Here is an example of utilizing a Chord object in Figure 4.



```
guitar_part.add(
  new Filter({
    key: new Chord("Cm7"),
    length: 4,
    active: (beat,measure)=>{
      return beat === 0 || beat === 8
    }
  })
)
```

Figure 4. Utilizing a Chord object

3. NOTATION

3.1 Musical Scales

We can write musical scales like "[scale][octave][#(optional)]". Available scales are from "C1" to "B4" for 4 octaves.

Instead of this notation, we can also use calculable one as "[index of scale]$[octave]". "1$1" corresponds to "C1" and "4$12" corresponds to "B4".

3.2 Chords

We can use general notation of chords as shown in Table 1. However, write directly "b" instead of "♭".

Table 1. Some examples of chord names

Chord Name	Root	Scheme
Cm7	C	m7
GbM9	Gb	M9
Am7(11)	A	m7(11)

4. TECHNICAL POINTS

4.1 Scheduling Sounds

The most basic function of precisely scheduling sounds is based on the idea described in the article of HTML5 Rocks [4]. While the context is playing, the context continuously collects chains of AudioNode objects from Part objects.

4.2 Considering Memory

Ongaq JS potentially uses a lot of memory, and we have worked hard to minimize its memory footprint. After repeated use, the program would sometimes crash, and we found that this was related to the number of allocated objects. By reducing object allocation to a minimum the program no longer experiences fatal performance problems.

5. FUTURE WORK

5.1 Enhancement

We plan to keep upgrading Ongaq JS. We would like to make it rich enough to use for live performance or interactive musical art.

Furthermore, in order to make the program more accessible for beginners, we plan to provide enough documents, samples and guidance for them to study by themselves.

5.2 Tools for STEM Education

We plan to develop visual programming tools for STEM education with Ongaq JS. In Japan, the government has decided to start lessons to cultivate basic skills of programming at all elementary schools in 2020. We want to provide suitable tools which can be applied to those needs.

As we can find at preceding projects, such as *Music Blocks* [5], music is really suitable for students to make friends with numbers, calculation or abstract concepts like variables or iteration. Therefore, we also plan to provide tools which encourage students learn programming and mathematics inclusively.

6. ACKNOWLEDGEMENTS

I would like to thank Torkel Berli from Wovn Technologies, Inc. for reviewing English usage, Hikaru Takakura, co-founder and engineer of CodeNinth Ltd., for taking part in discussions and code reviewing.

7. REFERENCES

[1] CodeNinth Ltd,. 2019. The GitHub repository of Ongaq JS. Retrieved from https://github.com/codeninth/ongaq-js

[2] CodeNinth Ltd,. 2019. The portal site of Ongaq JS. Retrieved from https://www.ongaqjs.com/

[3] Takakura, Hiroyuki. 2017. WebbyJam, a Web Tune Editor to Find Enjoyment. In *Proceedings of 3rd Web Audio Conference* (London, U.K, August 21 – 23, 2017). https://qmro.qmul.ac.uk/xmlui/handle/123456789/28066

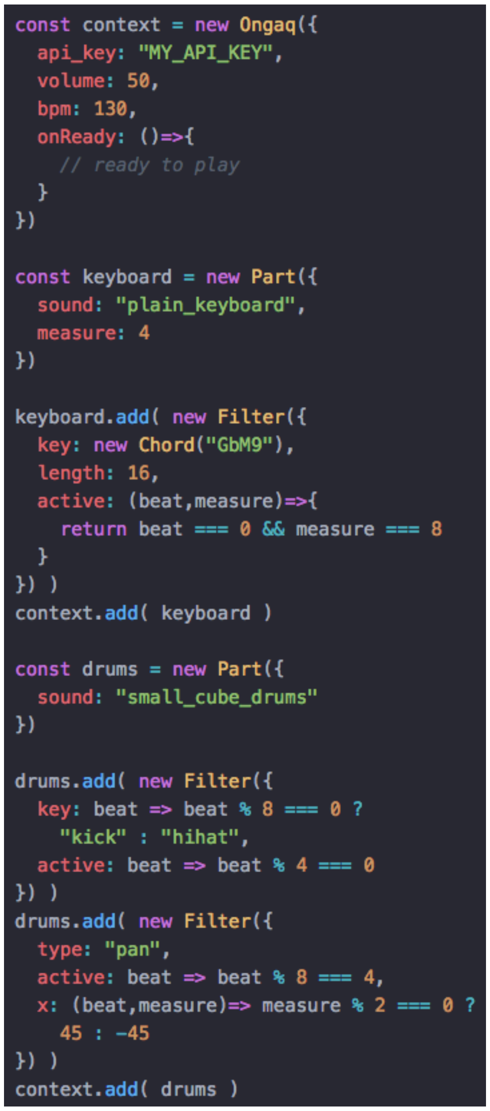
[4] Wilson, Chris. 2013. A Tale of Two Clocks - Scheduling Web Audio with Precision. Retrieved from https://www.html5rocks.com/en/tutorials/audio/scheduling/

[5] Sugar Labs. 2019. The GitHub repository of Music Blocks. Retrieved from https://github.com/sugarlabs/musicblocks

8. APPENDIX

8.1 Sample Code

This is a sample code that corresponds to a looping music consisting of drums and keyboard.



```
const context = new Ongaq({
  api_key: "MY_API_KEY",
  volume: 50,
  bpm: 130,
  onReady: ()=>{
    // ready to play
  }
})

const keyboard = new Part({
  sound: "plain_keyboard",
  measure: 4
})

keyboard.add( new Filter({
  key: new Chord("GbM9"),
  length: 16,
  active: (beat,measure)=>{
    return beat === 0 && measure === 8
  }
}) )
context.add( keyboard )

const drums = new Part({
  sound: "small_cube_drums"
})

drums.add( new Filter({
  key: beat => beat % 8 === 0 ?
    "kick" : "hihat",
  active: beat => beat % 4 === 0
}) )
drums.add( new Filter({
  type: "pan",
  active: beat => beat % 8 === 4,
  x: (beat,measure)=> measure % 2 === 0 ?
    45 : -45
}) )
context.add( drums )
```