New Opcodes for MIDI CC Preset Banks and MIDI Note-on Toggles for Csound in the *Bela*, Csound in the *Nebulae*, and Csound in General

Richard Boulanger[1] and John ffitch[2]

[1] Berklee College of Music
[2] Alta Sounds
`rboulanger@berklee.edu jpff@codemist.co.uk`

Abstract. In Csound, designing a MIDI synth that could store and recall MIDI Continuous Controller (CC) settings on the fly (MIDI presets), or turn a specific MIDI note-on message into an on/off toggle that would turn a Reverb, Flanger, or Distortion effect on or off are quite basic design needs that, until now, have required some pretty ingenious, advanced, and sometimes quite convoluted coding tricks to do the job. Some solutions use widgets and functions in *Cabbage*, *CsoundQt*, or *Blue*, but what if you `not` using Cabbage, CsoundQt or Blue, or what if you are not working on an OS, or in an Application, or on an embedded computing platform that supports them - such as the *Bela*, the *Qu-bit Nebulae*, or Chrome, Safari, Firefox? To address this general need, a new family of counting opcodes *cntCreate, cntCycles,cntDelete, cntRead, cntReset, cntState*, and a new family of MIDI controller opcodes *ctrlpreset, ctrlprint, ctrlsave, ctrlselect* have been added to Csound. In this paper, a discussion of their design, and examples of their use in general Csound and in Csound running on the *Bela* will be presented.

Keywords: Csound, MIDI, controller, preset, toggle

1 Introduction

Without using the widgets in Cabbage or CsoundQt, how does one save and recall MIDI controller settings in Csound? How do you use a MIDI note-on message, or an analog push button, to toggle an effect on and off on the Bela? These questions always come up when a beginner is designing their first MIDI instrument, or when they are trying to use Csound on the Bela to make a synth or guitar pedal. If one were using Cabbage or CsoundQt, the included widgets and functions make this an easy task; but we are not running Cabbage on the Bela (or on the Web), and not using the CsoundQt widgets in the Csound Web IDE. Surely, one could use *ctrlinit* to initialize one set of controllers, but how can you quickly save other setting while playing and moving the knobs, and how can one easily recall these settings? To address both of these 'preset' and 'toggle' problems, some inspiring 'opcode-only' solutions do appear in *The FLOSS Manual*, *The Csound Catalog*, and *The McCurdy Collection*, but these can be quite confusing to the beginner. Wouldn't it be nice if there were simple Csound opcodes that addressed these common needs in a simpler and more direct way? In this paper, we will present the new MIDI *ctrlpreset* family of opcodes and the new *cnt* counter family of opcodes. And we will show how they are used to store and recall banks of MIDI controllers (presets), and how the new counters can help to turn MIDI note-on messages and analog buttons into on/off latching toggles on the Bela, or on any host that runs Csound.

2 An Overview of the New Counter Family of Opcodes

A counter is an internal persistent object that carries a value which can be incremented by a given amount whenever the `count` opcode used. When the internal value reaches a maximum value it is reset to a minimum value. The internal state of range and increment can be retrieved by `cntState`. The counter can be reset to the initial state and can be read via `cntCycles`.

This family can be used in a variety of ways. As described in the next section, the simplest is as a toggle; but it could also count bars or particular events. [3]

3 Using *cntcreate* to Turn MIDI Notes into Toggles

When one presses (and releases) a key on a MIDI controller keyboard to play a note, one is actually sending two messages, a note-on message and a note-off message. Or, if MIDI running status is supported, when you release the note, you get another MIDI note-on message with a velocity of 0. In this case, each note you play can send two note on messages. As soon as you release the note, your "note-toggle" will turn the effect off. If you intend for your "toggle" to turn "on" when you play a note, and "stay-on" until you play the note again (to latch), then you need some way to count, keep track of the count, add and subtract from the count, reset the count. etc. Let's see how we do this in our example instrument:

```
<CsoundSynthesizer>

<CsInstruments>
gicnt cntCreate 1      ; for ASCII toggle
gicntNote cntCreate 1  ; to ignore MIDI note-off messages

instr 1
    kkey sensekey
    inote notnum
  if (kkey == 97) then ; the letter 'a'
        k1 count gicnt
```

[3] The current implementation doesn't allow for a varying increment, but that could give more flexibility to track mobile sounds for example.

```
    if k1==0 then
       event "i", 2, 0, -1
    else
       event "d", 2, 0, -1
    endif
   endif

  if (inote == 60) then ; the note 'c'
     i2 count_i gicntNote
   if i2==0 then
      event_i "i", 3, 0, -1
   else
      event_i "d", 3, 0, -1
   endif
  endif
endin

instr 2
  asig oscil .5, 440
      outall asig
 endin

instr 3
  asig oscil .5, 880
     outall asig
 endin
 </CsInstruments>

<CsScore>
i1  0  z
</CsScore>
</CsoundSynthesizer>
```

4 Saving and Recalling a Bank of Controller Presets

The new controller preset family of opcodes, *ctrlpreset, ctrlprint, ctrlprintpresets, ctrlsave,* and *ctrlselect* now provide the means for users to save and load presets to and from a file while running Csound. This will be shown in the example below. Here, two instruments in the following orchestra work to initialize a bank of presets, read them, and add to them. In the traditional way, the orchestra uses instrument 0 to initialize the controllers on MIDI channel 1 and essentially start off our FM instrument with an initial preset. Our *Preset_Bank* includes a bank of 12 user-created presets that respond to ASCII numbers 1 through 9 and letters a, b, and c. One can also use MIDI controller 28 to recall presets 1 - 12. These presets were originally written to the text file, *my_ctrlpresets.txt* with the *ctrlsave* opcode and were copied here. While the instrument is running, the *ctrlprintpresets* opcode displays the current preset file in the console and, most importantly, typing a capital A will appends a new preset to the current file, in effect, saving the current setting of all the active controllers.

```
<CsoundSynthesizer>

<CsInstruments>
 ctrlinit 1, 21,101, 22,11, 23,12, 24,16, 25,0, 26,82 ;Starting Preset

  ;to select presets 1-12, type 1 -> 9, or a, b, c from "instr FM"

instr Preset_Bank
 kpre1 ctrlpreset 0,1,21,101,22,94,23,01,24,06,25,14,26,02;ASCII 1
 kpre2 ctrlpreset 0,1,21,102,22,57,23,76,24,55,25,77,26,12;ASCII 2
 kpre3 ctrlpreset 0,1,21,103,22,12,23,13,24,16,25,84,26,22;ASCII 3
 kpre4 ctrlpreset 0,1,21,104,22,83,23,18,24,13,25,24,26,32;ASCII 4
 kpre5 ctrlpreset 0,1,21,105,22,20,23,48,24,33,25,94,26,42;ASCII 5
 kpre6 ctrlpreset 0,1,21,106,22,52,23,10,24,03,25,68,26,52;ASCII 6
 kpre7 ctrlpreset 0,1,21,107,22,09,23,02,24,07,25,01,26,02;ASCII 7
 kpre8 ctrlpreset 0,1,21,108,22,34,23,10,24,96,25,44,26,42;ASCII 8
 kpre9 ctrlpreset 0,1,21,109,22,04,23,91,24,66,25,04,26,92;ASCII 9
 kpre10 ctrlpreset 0,1,21,110,22,04,23,10,24,66,25,44,26,22;ASCII'a'
 kpre11 ctrlpreset 0,1,21,111,22,07,23,46,24,45,25,17,26,32;ASCII'b'
 kpre12 ctrlpreset 0,1,21,112,22,22,23,03,24,06,25,04,26,42;ASCII'c'

   ctrlprintpresets "./my_ctrlpresets.txt" ;print Presets to file
   ctrlprintpresets ;print bank in console with i "Preset_Bank" 0 .1

  turnoff
endin

instr  FM
   icps cpsmidi
   iamp ampmidi 0.6
   kc[] init 6
     kvol midic7  21, 0,1
     kcar midic7  22, 1,10
     kmod midic7  23, .1,10
     kndx midic7  24, 1,30
     kndx port    kndx,.1
     iatk midic7  25, .01,1
     irel midic7  26, .01,2

     kpre midic7  28, 1,12 ;use CC 28 to select from preset 1-12
 ktrig changed2 kpre
   if ktrig == 1 then
      kpre = int(kpre)
      ctrlselect kpre
      printk2 kpre
   endif

        asig foscil iamp, icps, kcar, kmod, kndx, 1
        kmgate linsegr 0, iatk, 1, irel, 0
        outs  (asig*kmgate)*kvol, (asig*kmgate)*kvol

 kc ctrlsave 1,21,22,23,24,25,26;MIDI Chan&CC vals to read & save
 kchar sensekey ;CC vals printed to console after playing next note

if kchar != 65 goto end0 ;ASCII char "65" is the letter 'A' (shift-a)
   ctrlprint kc ;prints Controller Settings (CC presets) to the console
   ctrlprint kc, "./my_ctrlpresets.txt" ;Appends CC settings to file

 end0:
  if kchar<49 || kchar>57 goto end1 ;ASCII numbers 1 -> 9
     kval = kchar - 48
     ctrlselect kval

 end1:
  if kchar <97 || kchar>122 goto end2 ;ASCII lower-case letters a -> z
    kval = (kchar - 97) + 10
    ctrlselect kval

 end2:
  if kchar != 86  goto end3 ;ASCII character "86" the letter 'V' (shift-v)
   kk ctrlpreset 0,kc ;add & number current state as a preset to the list
   ctrlprintpresets ;prints Controller Presets (CC presets) to console
   ctrlprintpresets "./my_ctrlpresets.txt" ;appends CC presets to a file

 end3:
endin
</CsInstruments>

<CsScore>
 f0 3333
 f1 0 8192 10 1
i "CC_Preset_Bank"  0  0.1
</CsScore>
</CsoundSynthesizer>
```

5 Conclusion

The motivation for the design of these counter and controller opcodes was to simplify the design and enhance the functionality of Csound-based systems built for the Bela and the Qu-bit Nebulae, and they have more than met the hopes and expectations of Csounders working on these platforms. For years now, it has been empowering to be able to build Csound instruments that would respond to MIDI and change in response to MIDI controllers; but, it has been tedious and difficult to save controller presets, while tweaking them and playing the instruments from a MIDI controller. Building instruments with the CsoundQt widgets was one solution; but you were tied to running the instruments from a computer running CsoundQt. Cabbage has buttons, switches, and toggle widgets, and includes functions for saving (and naming) presets. And, it is worth noting that one of the greatest assets of working with Csound in Cabbage is that one can export these ”instruments” and ”effects” as Audio Units or VST plugins, and stand-alone applications. But still, Cabbage does not run on the Bela or on the Nebulae or on the Web. And so, this new opcode family to support the saving, storing, numbering, and recalling of MIDI continuous controller presets is a great addition to Csound as it brings these capabilities to the designer and supports their exportation to any platform that runs Csound. The authors hope that you would agree with them that it can be (and is) really great to add this functionality to the huge collection of MIDI-based Csound instruments in the Csound Catalog, and especially to play them, fine-tune them, and store them as you go, and, moreover, to recall them after you are done your ’patch-design’ session.