Celluloid

A Language Tutorial

Charlie Robbins
Sean McDuffee
Dan Federman
David Flerlage
Blake Arnold

Introduction:

Celluloid is a language designed to simplify the coordination of different types of media inputs, such as audio files, video files, midi-devices, onto different types of outputs, such as speakers, displays, lights, etc. Celluloid provides several unique features to help in this process including timelines, constraints, and events.

This tutorial is intended to be an informal introduction to the nuances of Celluloid. The tutorial begins with a simple example and then progressively builds on previous examples until a complicated program is constructed that illustrates most of the functionality of the Celluloid language. For a complete description of the functionality of Celluloid, please see the Celluloid Reference Manual. A number of interesting features are not fully enumerated here. In particular, a powerful feature of Celluloid that is not described in this tutorial is the ability to extend some of the basic types, such as devices, events, and constraints.

Chapter 1 - A simple Celluloid program:

The most basic Celluloid program consists of a single input, a single output, and a single timeline. Before use, all objects must be declared, such as:

```
timeline timeline1
```

And all input and output devices must be instantiated, as in the following two lines.

```
input audio1 = new AudioFile(*some file*)
output output1 = nˆew Audio(*some device key*)
```

Here *audio1* is not actually an input device but rather a file. Bringing in an audio file like in the example just requires specifying the file’s pathname. The output *output1* is specified as an audio device. Bringing in the audio device requires the keyword mapped from a user-defined *config.io* file. Specifically, each line in the *config.io* is a mapping of user specified key to a device location on the target machine. To use a device that is described in the *config.io* file, the programmer uses the user-defined key like a variable name. More on *config.io* later in the tutorial.

In order to use any sort of input, it must first be added to a timeline. To perform operations on an input, a constraint function must specify what the action should be and at what time the action should be executed. In the following, play is a constraint function that operates on the input *audio1* with the time parameter *@start*. The code between the *do..end* block specifies the actions on the timeline.

```
in timeline1 do
  play audio1 @start
end
```

Simply adding the input devices to a timeline in some specified order is not enough to start playback. The execution of a timeline does not start until they are added to an output. In the following line, *timeline1* is set to play on *output1* at the start of the program.

```
in output1 do
  play timeline1 @start
end
```

This simple program has taken one input, attached it to a timeline, and then outputted that timeline to a device. In the following examples we will build on these ideas as we introduce more of what the Celluloid language has to offer.

Chapter 2 - Basic media sequencing:

Celluloid allows for simple sequencing of multiple inputs such as audio files. Here we show how two audio files can be sequenced together on a single timeline and then outputted just as before. Once again, timelines must be declared, and all input and output devices must be declared and instantiated before use:

```
timeline timeline1

input audio1 = new AudioFile(*some file*)
input audio2 = new AudioFile(*some file*)
output output1 = new Audio(*some device key*)
```

Like the previous example, we operate on the inputs within an *in..do..end* block to build up the timeline. As shown below, the built-in functions *pause* and *stop* can be used in the same manner as *play* to designate when on the timeline the functions apply. In addition to the time literal *@start*, we can designate time directly as in *@1m* to mean "at 1 minute".

```
in timeline1 do
  play audio1 @start
  pause audio1 @1m
  play audio2 @1m
  stop audio2 @2m
  play audio1 @2m
end
```

This block created an audio sequence with the first audio file playing for 1 minute from its start then the second audio file playing for an additional minute from that point.

Note that the *@start* in the line *play audio1 @start* is referring to the start of the timeline, not the file. The *play* function plays a file from the place it was last paused. There are two special cases in which the *play* function will start playing a file from the beginning: the first time *play* is called, and the next time *play* is called after the function *stop* has been called on the input. Continuous input devices such as a midi controller can not have *pause* and *stop* called on them and a compiler error will result (see section 6.3 of the Celluloid Reference Manual for more details). For these inputs the function *play* just attaches the current stream from the device to the timeline.

Our second program connects to the output in exactly the same way as before.

```
in output1 do
  play timeline1 @start
end
```

As we have shown, it's very easy and straightforward to create audio, and video for that matter, sequences in Celluloid.

Chapter 3 - Introducing constraints:

We now move on and introduce the use of constraints to composite an output with four video files playing in each corner of a display.

```
timeline timeline1

input video1 = new VideoFile(*some file*)
input video2 = new VideoFile(*some file*)
input video3 = new VideoFile(*some file*)
input video4 = new VideoFile(*some file*)
output output1 = new Video(*some display*)
```

One of the built-in constraints in Celluloid is size. Constraints define constraint functions. A constraint function is given a target to be applied to and a comma-separated parameter list unique to the function. The constraint size takes four parameters, the x-y positions of the top left and bottom right corners of the output device. The numbers are given between 0 and 1, so the point 0.5, 0.5 is the center of the output device. The point 0,0 is the upper left corner with the point 1,1 corresponding to the bottom left corner. The keyword *all* can be used with constraint functions such as *play* to apply the function to all inputs that have been referenced in the *in* block.

```
in timeline1 do
  size video1 0, 0, .5, 0
  size video2 .5, 0, 1, 0
  size video3 0, .5, .5, 1
  size video4 .5, .5, 1, 1
  play all @start
end
```

```
in output1 do
  play timeline1 @start
end
```

So we've brought in four video files, which could easily have been four live video streams from a camera, in and played them out on a 2x2 grid with very little code. Let's now expand on this example with more of the functionality Celluloid offers.

Chapter 4 - Looping & when keyword:

Taking the previous example, we can now build a simple program for a closed-circuit television security display. Instead of inputting from a set of video files, we input from a set of video devices that have been set up in a *config.io* file.

```
timeline timeline1

input video1 = new Video(*some video device key*)
input video2 = new Video(*some video device key*)
input video3 = new Video(*some video device key*)
input video4 = new Video(*some video device key*)
input video5 = new Video(*some video device key*)
input video6 = new Video(*some video device key*)
input video7 = new Video(*some video device key*)
input video8 = new Video(*some video device key*)

output output1 = new Video(*some display*)
```

Once again we set size constraints of for each video to the output. Here, we set two video streams to each quadrant of the output display device.

```
in timeline1 do
  size video1 0, 0, .5, 0
  size video2 .5, 0, 1, 0
  size video3 0, .5, .5, 1
  size video4 .5, .5, 1, 1

  play all @start

  size video5 0, 0, .5, 0
  size video6 .5, 0, 1, 0
  size video7 0, .5, .5, 1
  size video8 .5, .5, 1, 1
  # continued below...
```

Note the use of *play all* here. The use of the keyword *all* only applies to all inputs that have been referenced so far in the *in* block.

The next section of code in the *every* block sets up 10 second video loops for each quadrant of the eventual output device. In order to do so we introduce the keywords *every* and *when*. The keyword every takes a time literal and then does what is put in a *do..end* block every time that literal evaluates. The *every* keyword can also have a *when* keyword after the time specification that can check for events like *finished* or can check the truth of attached flags on an input that are set by functions. The flag *isPlaying* is set by the *play* function in the line *play all @start* above.

The following code also introduces the syntax for commenting Celluloid code by starting a line with the hash symbol #. Anything written after a #-symbol will not be evaluated.

```
# continued from above...
# quadrant 1
every @10s when video1.isPlaying do
  play video5 @now
end
every @10s when video5.isPlaying do
  play video1 @now
end

#quadrant 2
every @10s when video2.isPlaying do
  play video6 @now
end
```

```
  every @10s when video6.isPlaying do
    play video2 @now
  end

  #quadrant 3
  every @10s when video3.isPlaying do
    play video7 @now
  end
  every @10s when video7.isPlaying do
    play video3 @now
  end

  #quadrant 4
  every @10s when video4.isPlaying do
    play video8 @now
  end
  every @10s when video8.isPlaying do
    play video4 @now
  end
end

in output1 do
  play timeline1 @start
end
```

There we have it! We've put together a very straightforward program for creating a security station display with very little coding.

Chapter 5 - Specifying config.io:

To bridge the gap between Celluloid and the underlying operating system, the user must create a special file that maps each device's location to a unique key that can be used inside Celluloid code to refer to that specific device. For a detailed description, please see section 3.3 of the Celluloid Reference Manual. To create a device file, each line requires three separate string tokens: a key that will be used within the source code, the type of device, and the physical location of that device on the operating system.

Example config.io file:

```
#<key>          <type>      <location>
Monitor         DVI         0x0863
MIDIdevice      MIDI        0x24400000
Microphone      MIC         0x10DECB79
```