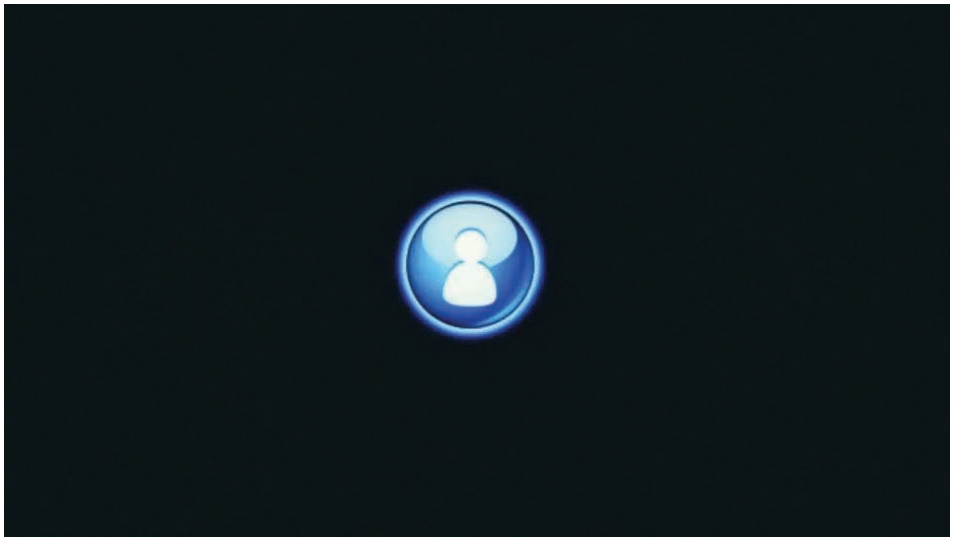
Hosts and Modules

Edward A. Lee

Programming the Swarm Workshop

Berkeley, CA

May 27-29, 2015



Sponsored by the TerraSwarm Research Center, one of six centers administered by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

We are Here

https://www.terraswarm.org/accessors/wiki/Main/ProgrammingTheSwarmWorkshopAgenda

View | Edit | History | Attach | Print

TerraSwarm

Navigation

- TerraSwarm Home
- Accessors Home
- Accessors Wiki
- Editing Instructions

edit SideBar

Search

Go

Programming The Swarm Workshop Agenda

May 27-29, 2015, Berkeley.

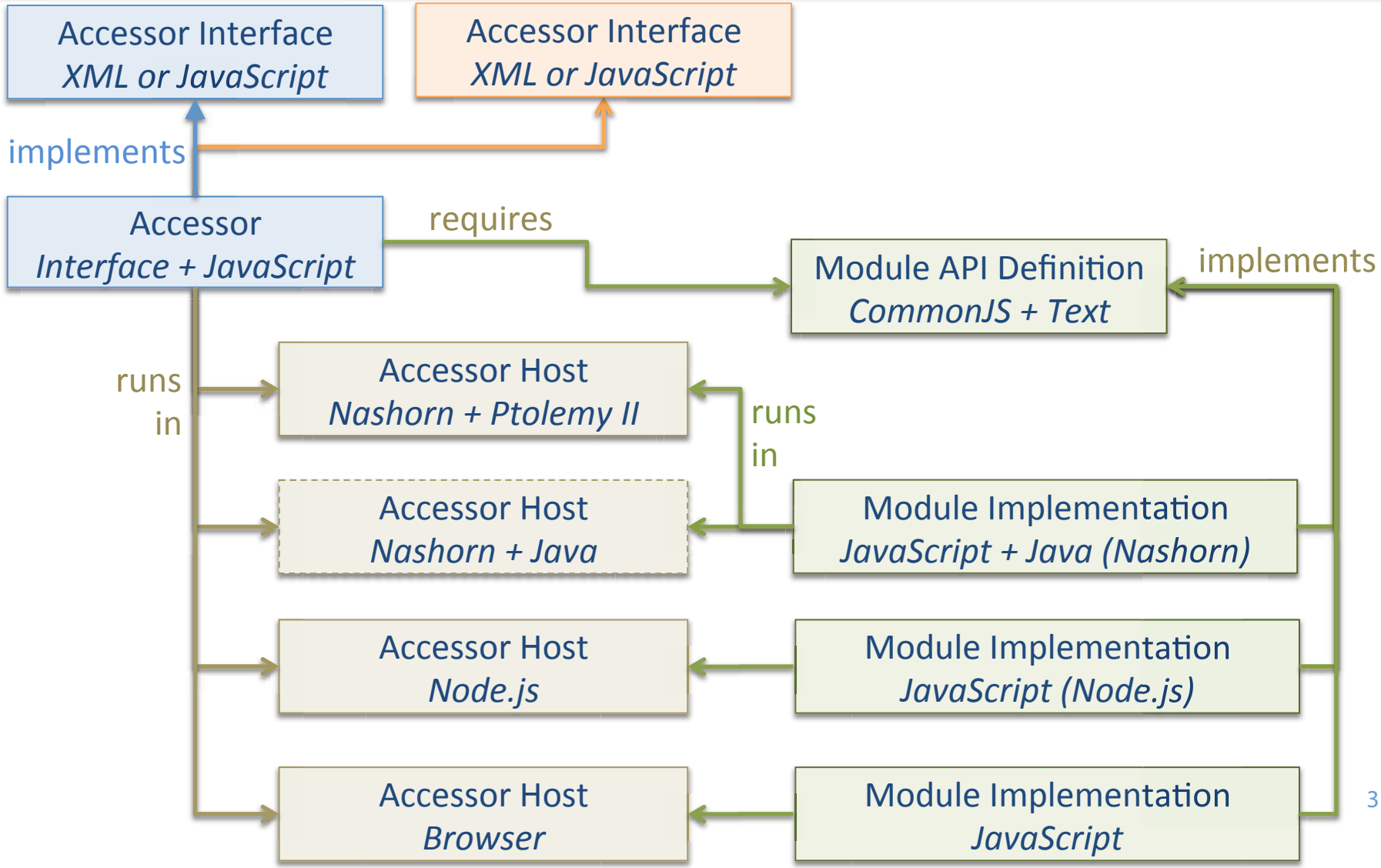
Tentative Agenda. We expect the agenda to be very fluid, so this is just a guideline.

Important: Do the homework. In particular, install Ptolemy II. If you fail to do this, we will try to h

Wednesday May 27:

8:00am	* For those who d o the homework: Software installation workshop (Christopher) * For those who d homework: Breakfast.
9:00am	Introduction and zation (15 minutes)
9:15am	Accessor Design rd Lee, 30 minutes)
9:45am	Common Exercis d an Audio Accessor, parts 1 and 2 (45 minutes)
10:30am	Break (15 minutes)
10:45am	Nashorn/Ptolemy II host (Edward Lee, 15 minutes)
11:00am	JavaScript Functions and Modules (Edward Lee, 30 minutes)
11:30am	Common Exercise: Build an Audio Accessor, parts 3 and 4 (45 minutes)

Accessor Architecture Version 0.1a



JavaScript Ecosystem

- JSON (JavaScript Object Notation):
  - Textual notation for arrays and key-value pairs.
- JavaScript engines:
  - Browsers
  - Node.js (Google's V8 / Chrome JavaScript in C++)
  - Nashorn (Java implementation of JavaScript)
- JavaScript APIs
  - Vert.x (Eclipse foundation application platform)
  - Avatar (Oracle's Node.js on Nashorn)

JavaScript Ecosystem

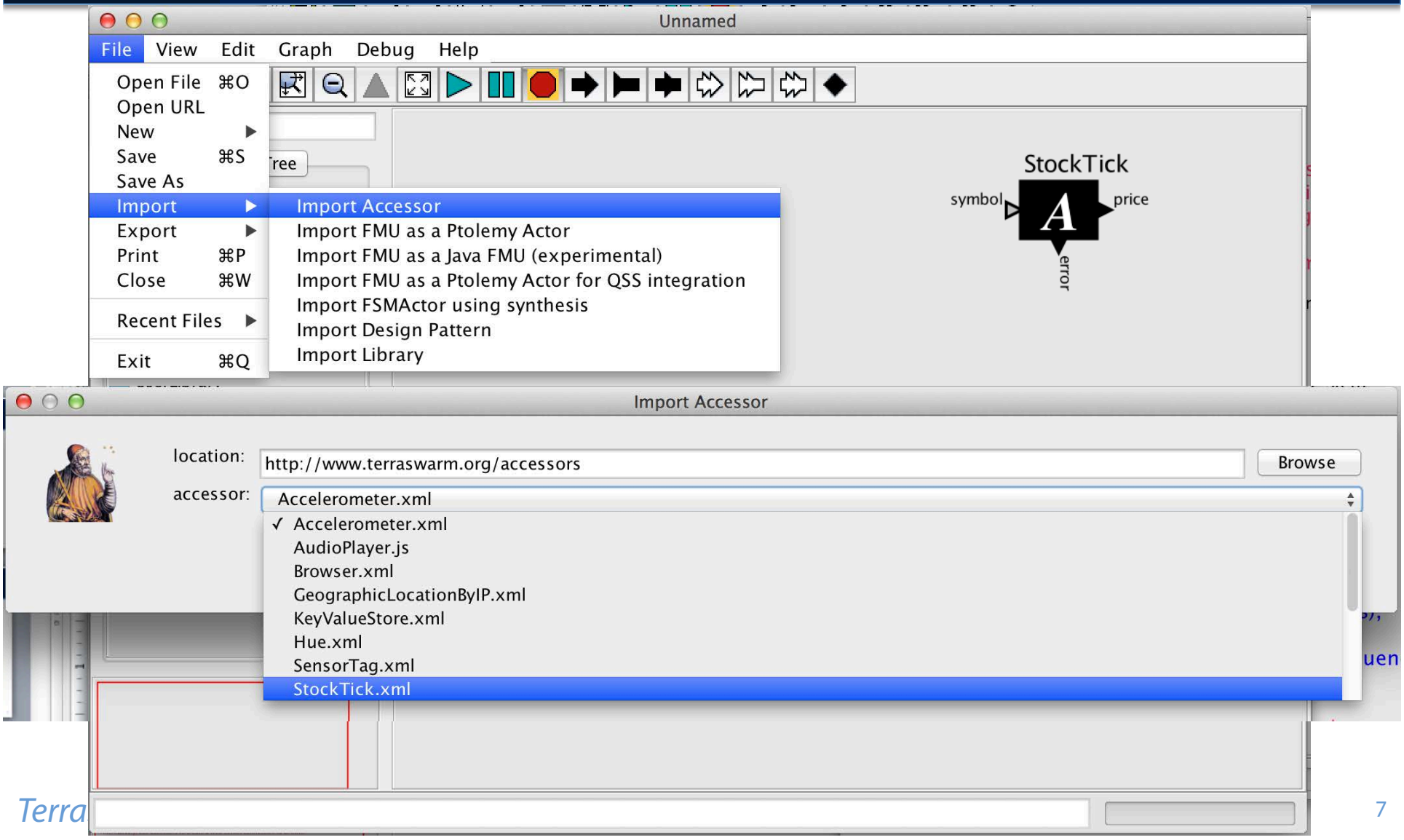
- Browsers
  - Extensive support for manipulating documents
  - Limited support for network access (e.g., AJAX and XMLHttpRequest object)
- Node.js
  - Intended for “server side” (not in a browser)
  - Rich, general-purpose application platform
  - C++ on x86, ARM, MIPS, w/ OSX, Windows, Linux
- Nashorn
  - Java 8’s JavaScript engine
  - With Vert.x, a rich, general-purpose application platform
  - Runs on the JVM with full access to Java.

A Swarmlet Host

- Can instantiate and execute accessors
- Provides a JavaScript engine
- Implements required top-level functions
- Provides required modules
- Provides zero or more optional modules

- Provides composition of accessors

Ptolemy II Swarmlet Host
Can instantiate accessors



A Swarmlet Host

- Can instantiate and execute accessors
- Provides a JavaScript engine
- Implements required top-level functions
- Provides
- Provides

In Ptolemy II, this is Nashorn, with full access to Java and Vert.x. Security? Needs to be sandboxed…

- Provides composition of accessors

A Swarmlet Host

- Can instantiate and execute accessors
- Provides a JavaScript engine
- Implements required top-level functions
- Provides required modules
- Provides zero or more optional modules

- Provides composition of accessors

Top-Level Java Script Functions

This page describes the top-level JavaScript functions required to be provided by an accessor host in the version 0.1a (Berkeley) of the accessor specification. See Top-Level JavaScript Functions B for the Michigan version.

The following functions, listed alphabetically, enable the script to get inputs and produce outputs, and also provide a small set of utility functions.

- **addInputHandler**(*function*, *input*): Specify a function to invoke when the input with name *input* (a string) receives a new input value. Note that after that function is invoked, the accessor's **fire**() function will also be invoked, if it is defined. If the specified function is null, then only the **fire**() function will be invoked. If the *input* argument is null or omitted, then the specified function will be invoked when *any* new input arrives to the accessor. This function returns a handle that can be used to call **removeInputHandler**(). **Important:** If you add an input handler during execution of the swarmlet, e.g. in initialize(), then you must call removeInputHandler(), or your input handler will persist across executions of the swarmlet. Do this in wrapup().
- **alert**(*message*): Pop up a dialog with the specified *message*.
- **clearInterval**(*handle*): Clear a timer interval action with the specified *handle* (see **setInterval**()).
- **clearTimeout**(*handle*): Clear a timeout with the specified *handle* (see **setTimeout**()).
- **get**(*input*): Get the value of an input with name *input* (a string). See Input.
- **print**(*message*): Print the specified *message* to the console (standard out).
- **removeInputHandler**(*handle*, *input*): Remove the callback function with the specified handle (returned by **addInputHandler**()) for the specified *input* (a string).
- **require**(*moduleName*): Load the specified module by name (a string) and return a reference to the module. The reference can be used to invoke any functions, constructors, or variables exported by the module (see Module Specification).
- **send**(*value*, *name*): Send a *value* to an input or output with the specified *name* (a string). See Input and Output.
- **setInterval**(*function*, *milliseconds*): Set the specified *function* to execute after specified time in milliseconds and again at multiples of that time, and return a handle. The specified function may send data to outputs or inputs and may get data from inputs. If additional arguments are provided beyond the first two, then those arguments are passed to the function when it is called.
- **setTimeout**(*function*, *milliseconds*): Set the specified *function* to execute after specified time in milliseconds and return a handle. The specified function may send data to outputs or inputs. If additional arguments are provided beyond the first two, then those arguments are passed to the function when it is called.

A Swarmlet Host

- Can instantiate and execute accessors
- Provides a JavaScript engine
- Implements required top-level functions
- Provides required modules

Built-In Java Script Modules

The following objects provide bundles of functions and should be built in to any accessor host.

- console: Provides various utilities for formatting and displaying data.
- events: Provides an event emitter design pattern (requires util).
- util: Provides various utility functions.

- Ca
- Pr
- Im
- Pr
- Pr
- Pr

Optional Java Script Modules

The following objects provide bundles of functions. An accessor that uses one or more of these modules must declare that requirement using the require tag or JavaScript function.

Reasonably Well-Developed Modules

- eventbus: Provide publish and subscribe on a local network through a Vert.x event bus.
- httpClient: Provide support for HTTP clients.
- webSocket: Provide full-duplex we[illegible]

Unfinished Modules

- audio: Provide access to the host
- browser: Provide display in the de
- discovery: Provide device discove
- localStorage: Provide persistent k
- mqtt: Provide support for MQTT p
- obd
- serial
- coap
- rabbitmq
- ble

To implement a module, see the Module Specification.

Audio

This package provides access to the host's audio hardware.

Functions

- **Player**(*options*): Constructor for an audio player. This acquires access to the audio hardware. FIXME: Currently, no options are supported. The returned object provides the following functions:
  - **play**(*data*): Play back the specified array of numbers, which give audio samples.
  - **stop**(): Stop audio playback. After calling this, you will need a new instance of Player to resume playback.

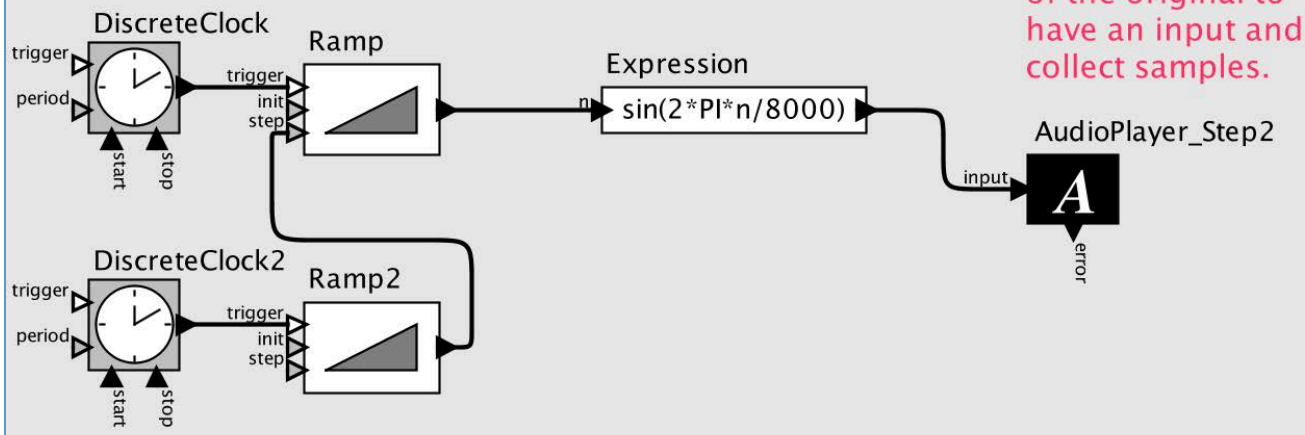
Solution to Step 2

Checked in to
$PTII/
org/
terraswarm/
accessor/
demo/
Audio/
solutions/
models/
Audio_Step2.xml

Step 2 Solution

At the right is an instance of the AudioPlayer accessor.
In its original (incomplete) form for this exercise, it has no input port. It just plays a fixed sinusoidal sound when it is initialized. You should augment the accessor definition to endow it with an input, and then plug in in place of the Plotter below to generate a chirp signal.

The accessor below is a modification of the original to have an input and collect samples.



DE Director

The composition semantics in this model is DE (discrete events), where components send each other time-stamped events.
The parameters of the Clock and Ramp actors control the frequency of the sinusoidal signal that is generated.

Extra credit: Replace the lower Ramp with some other accessor, such as StockTick, to control the frequency of the signal based on stock prices. Alternatively, use accessors to create a web socket connection to another machine to control the frequency from another machine.

Step 2 Solution

This accessor is designed to run on any host that implements the “audio” module.

In principle, all the hosts should be able to do this, though we’ve only implemented it on Ptolemy II/Nashorn. Project idea?

```
5  exports.setup = function() {
6      accessor.author('Edward A. Lee');
7      accessor.version('0.1 $Date:$');
8      accessor.input('input');
9  };
10
11 var audio = require("audio");
12
13 // State variables for this accessor:
14 var buffer = [];
15 var sampleCount = 0;
16 var handle = null;
17 var player = null;
18
19 exports.initialize = function() {
20     sampleCount = 0;
21     player = new audio.Player();
22     handle = addInputHandler(handleInput, 'input');
23 }
24
25 function handleInput() {
26     var sample = get('input');
27     buffer[sampleCount++] = sample;
28     if (sampleCount == 128) {
29         player.play(buffer);
30         sampleCount = 0;
31     }
32 }
33
34 exports.wrapup = function() {
35     if (player != null) {
36         player.stop();
37         player = null;
38     }
39     if (handle == null) {
40         removeInputHandler(handle, 'input');
41     }
42 }
```

audio module, Ptolemy II/Nashorn implementation

$PTII/
ptolemy/
actor/
lib/
jjs/
modules/
audio/
audio.js

```
1   // CommonJS module to access audio hardware on the host.
2   // Reference to the Java class documented at:
3   //    http://terra.eecs.berkeley.edu:8080/job/ptII/
4   //         javadoc/ptolemy/media/javasound/LiveSound.html
5
6   var LiveSound = Java.type('ptolemy.media.javasound.LiveSound');
7
8   exports.Player = function(options) {
9       LiveSound.setSampleRate(8000);
10      LiveSound.startPlayback(this);
11  }
12
13  exports.Player.prototype.play = function(data) {
14      // NOTE: Convert array into 2-D array required by LiveSound.
15      LiveSound.putSamples(this, [data]);
16  }
17
18  exports.Player.prototype.stop = function() {
19      LiveSound.stopPlayback(this);
20  }
```

Any CommonJS module found in this modules directory is supported by the Ptolemy II host.

Defining a CommonJS Module

$PTII/
ptolemy/
actor/
lib/
jjs/
modules/
audio/
package.json

```
{
  "name": "audio",
  "version": "0.1.0",
  "main": "audio.js",
  "license" : "BSD-3-Clause",
  "contributors": [
    {"name" : "Edward A. Lee"
    , "email" : "eal@eecs.berkeley.edu"}
  ],
  "description": "A module for accessing audio hardware on the host."
}
```

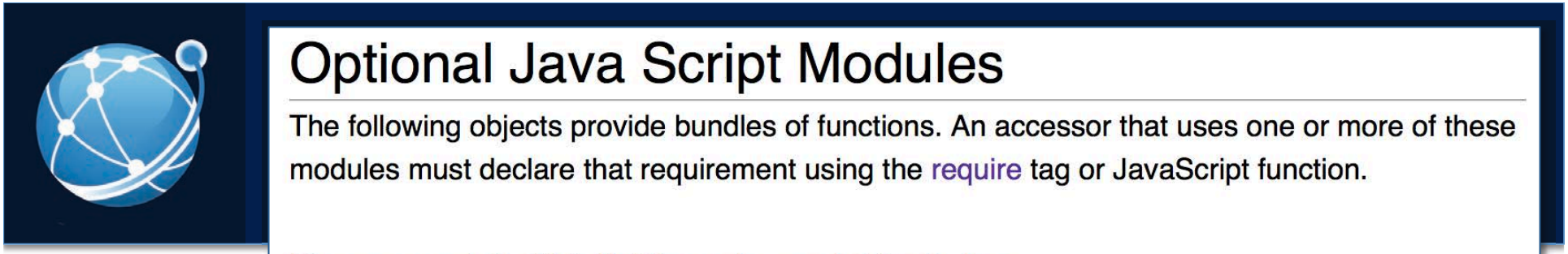
package.json file makes this a CommonJS module.

audio module, Ptolemy II/Nashorn implementation

$PTII/
ptolemy/
actor/
lib/
jjs/
modules/
audio/
audio.js

```
1   // CommonJS module to access audio hardware on the host.
2   // Reference to the Java class documented at:
3   //    http://terra.eecs.berkeley.edu:8080/job/ptII/
4   //         javadoc/ptolemy/media/javasound/LiveSound.html
5
6   var LiveSound = Java.type('ptolemy.media.javasound.LiveSound');
7
8   exports.Player = function(options) {
9       LiveSound.setSampleRate(8000);
10      LiveSound.startPlayback(this);
11  }
12
13  exports.Player.prototype.play = function(data)
14      // NOTE: Convert array into 2-D array requi    by LiveSound.
15      LiveSound.putSamples(this, [data]);
16  }
17
18  exports.Player.prototype.stop = function() {
19      LiveSound.stopPlayback(this);
20  }
```

This implementation uses a utility class in Ptolemy II, implemented in Java, leveraging Nashorn's Java interface.



Optional Java Script Modules

The following objects provide bundles of functions. An accessor that uses one or more of these modules must declare that requirement using the require tag or JavaScript function.

Reasonably Well-Developed Modules

- eventbus: Provide publish and subscribe on a local network through a Vert.x event bus.
- httpClient: Provide support for HTTP clients.
- webSocket: Provide full-duplex web socket interfaces and functions for web socket servers.

Unfinished Modules

- audio: Provide access to the host audio hardware.
- browser: Provide display in the default browser.
- discovery: Provide device discovery for devices on the local area network.
- localStorage: Provide persistent key-value storage based on local files.
- mqtt: Provide support for MQTT protocol clients.
- obd
- serial
- coap
- rabbitmq
- ble

More modules

See: https://chess.eecs.berkeley.edu/ptexternal/src/ptII/doc/codeDoc/js/index.html

A Swarmlet Host

- Can instantiate and execute accessors
- Provides a JavaScript engine
- Implements required top-level functions
- Provides
- Provides

In Ptolemy II, this is governed by a *director*. The DE Director seems to work rather nicely with accessors.

- Provides composition of accessors

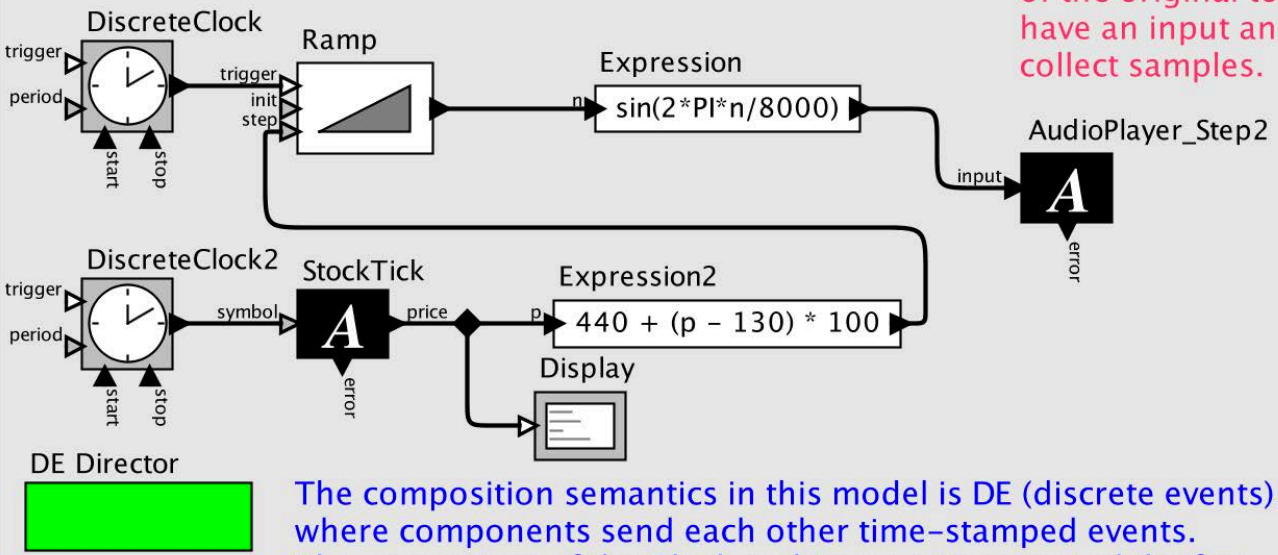
Solution to Step 2 – Extra Credit

Checked in to
$PTII/
org/
terraswarm/
accessor/
demo/
Audio/
solutions/
models/
Audio_Step2_
ExtraCredit.xml

Step 2 Extra Credit Solution

At the right is an instance of the AudioPlayer accessor.
In its original (incomplete) form for this exercise, it has no input port. It just plays a fixed sinusoidal sound when it is initialized. You should augment the accessor definition to endow it with an input, and then plug in in place of the Plotter below to generate a chirp signal.

The accessor below is a modification of the original to have an input and collect samples.



The composition semantics in this model is DE (discrete events), where components send each other time-stamped events.
The parameters of the Clock and Ramp actors control the frequency of the sinusoidal signal that is generated.

Extra credit: Here, the stock price of Apple is retrieved and compared against a nominal price of 130. If it is trading at the nominal price, the output will be a sinsoid with frequency 440 Hz (middle A on the piano). Above that price yields a higher pitch, and below that price yields a lower pitch.

Common Exercise – Part 3

3. Modify the CommonJS audio module.

An accessor is designed to be executable by any accessor host, not just Ptolemy II/Nashorn. Many accessors, including AudioPlayer, require that the host provide some capability. For AudioPlayer, the host needs to provide access to the audio hardware of machine. This requirement is expressed in the accessor by the line

```
var audio = require("audio");
```

This line refers to a **module** called "audio". Every accessor host that is capable of hosting AudioPlayer must provide an implementation of that module.

...

Your task now is to augment the module by adding one more object type, Capture, with two functions, **get** and **stop** that retrieve an array of audio samples and stop the capture, respectively. We suggest you make these modifications directly in the audio.js file using an editor of your choice.

Common Exercise – Part 3

You can experiment with your module by instantiating an actor called JavaScript in a new Ptolemy II model. We suggest placing a DE Director in the model, and then double clicking on the JavaScript actor and writing your test code as in the following example:

```
var audio = require("audio");
exports.initialize = function() {
  var capture = new audio.Capture();
  var data = capture.get();
  for each (sample in data) {
    print(sample);
  }
  capture.stop();
}
```

Now, each time you run the model, the body of code in your initialize function will execute.

Common Exercise – Part 4

4. Create an AudioCapture accessor.

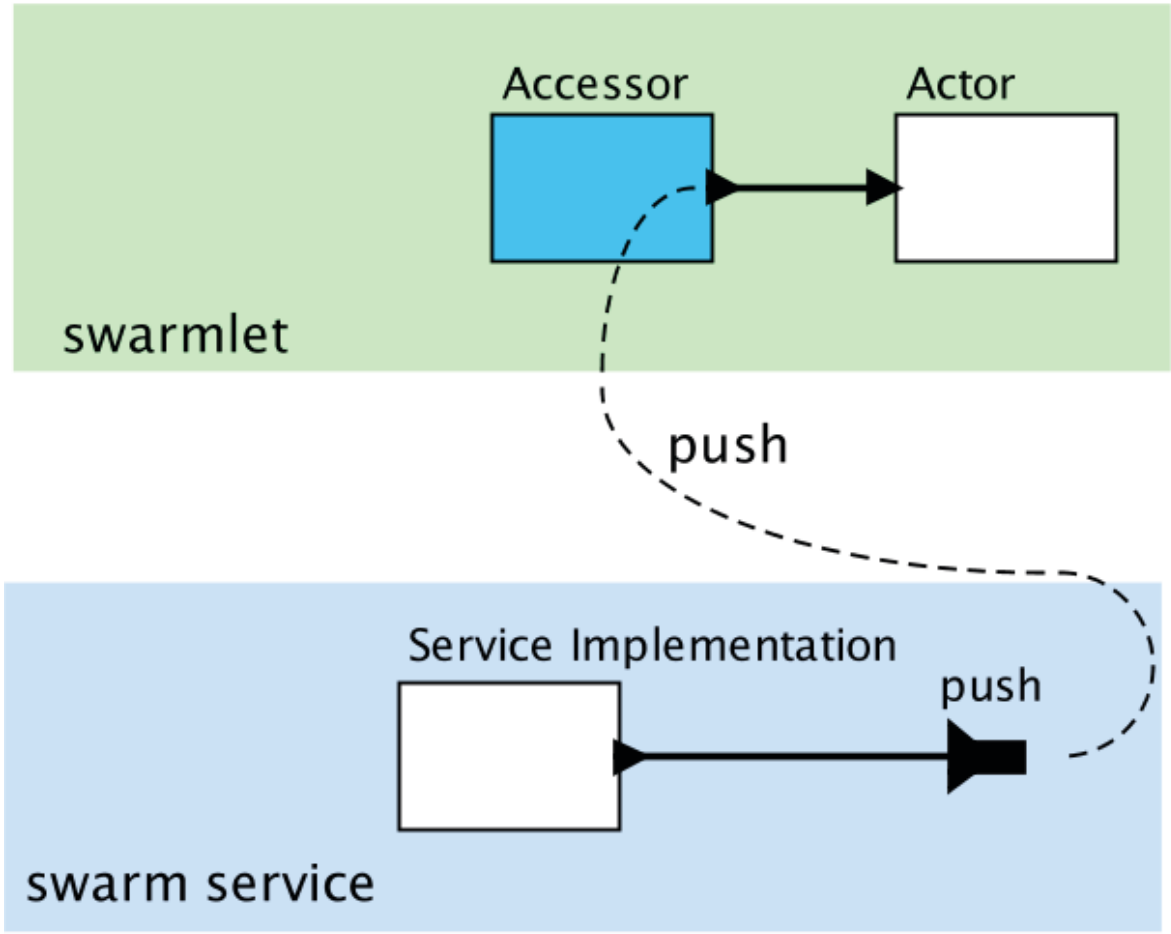
Your final task is to create an AudioCapture accessor. In the same directory where you downloaded and modified AudioPlayer, create a new file AudioCapture.js that defines this accessor. To import this accessor to Ptolemy II, you will need to create a third file called **index.json** that contains an array of accessor definition files provided in this directory, like this:

```
[ 'AudioPlayer.js', 'AudioCapture.js' ]
```

The index.json file defines an **accessor library**. Once you have created this file, in vergil, you can invoke File->Import->Import Accessor to instantiate accessors from your new accessor library.

Horizontal Contracts

An AudioCapture accessor is a *spontaneous source*, producing data when it is available. How to control and time its execution?



Blocking? Threading? Timing?

OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

ptolemy.media.javasound

Class LiveSound

java.lang.Object
    ptolemy.media.javasound.LiveSound

```
public class LiveSound
extends java.lang.Object
```

This class supports live capture and playback of audio samples. For audio capture, audio samples are captured from the audio input port of the computer. The audio input port is typically associated with the line-in port, microphone-in port, or cdrom audio-in port. For audio playback, audio samples are written to the audio output port. The audio output port is typically associated with the headphones jack or the internal speaker of the computer.

LiveSound.getSamples() blocks until enough samples are ready. What are the consequences on a swarmlet? How should you handle this?