TerraSwarm

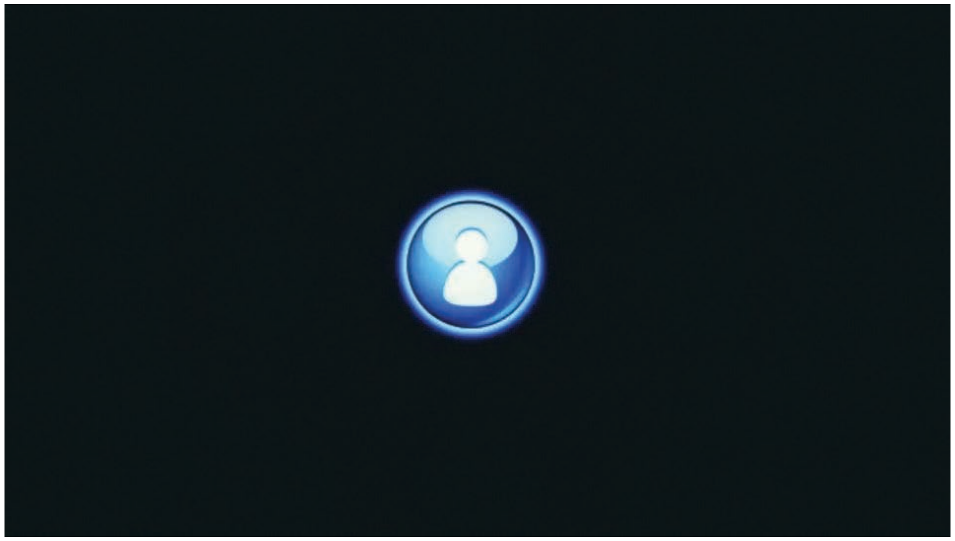
Accessor Design

Edward A. Lee

Programming the Swarm Workshop

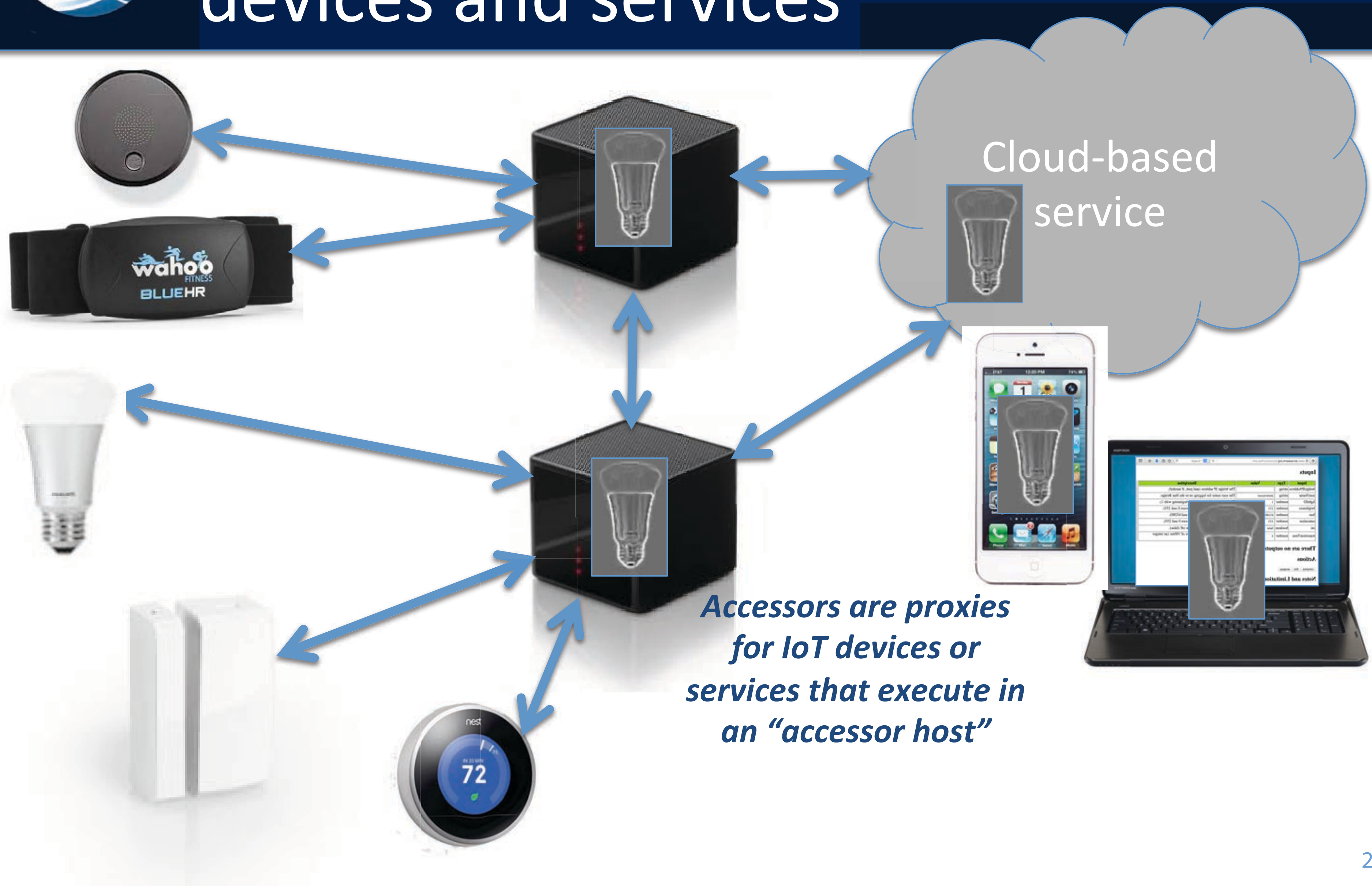
Berkeley, CA

May 27-29, 2015



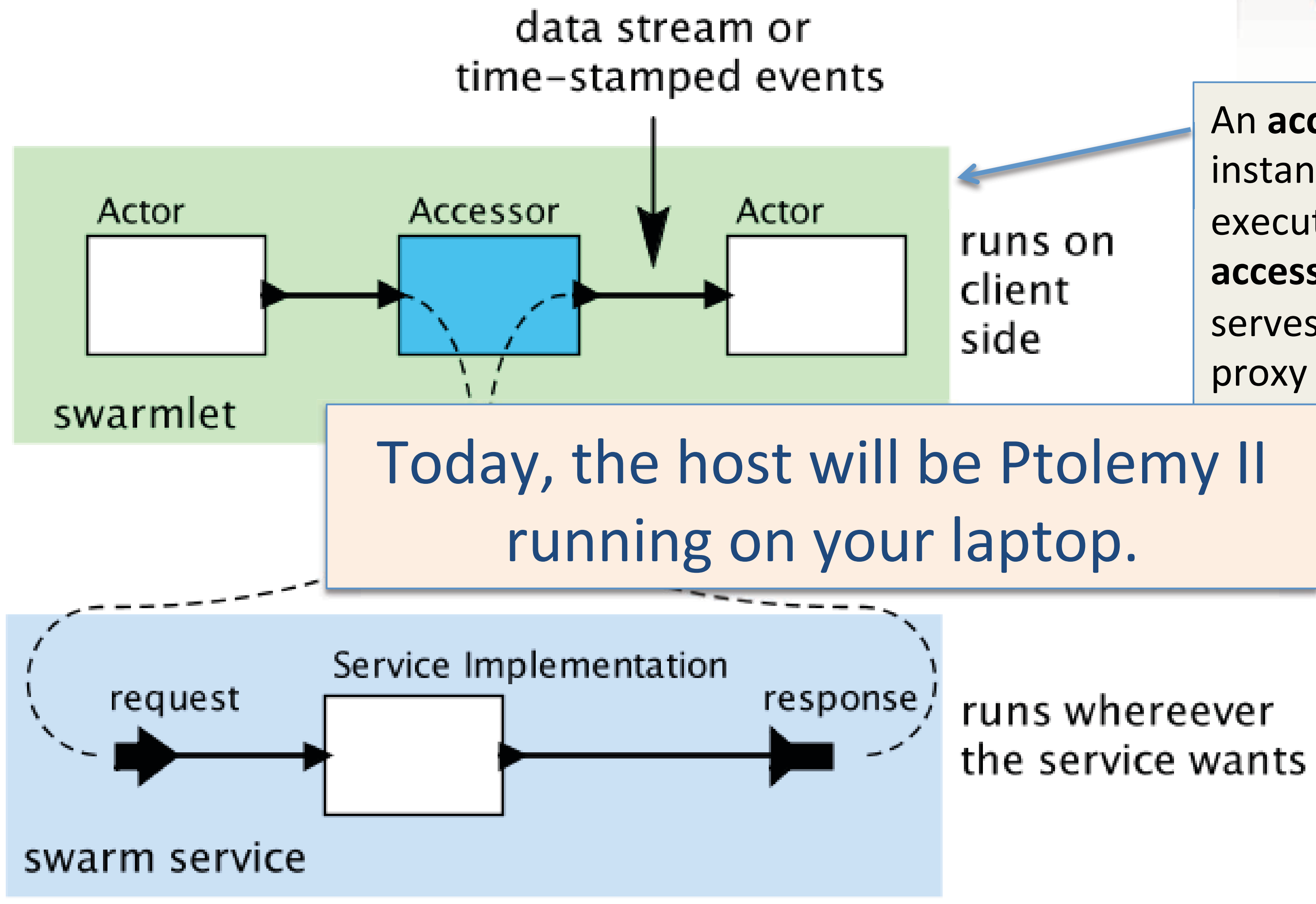
Sponsored by the TerraSwarm Research Center, one of six centers administered by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

Accessors are local proxies for devices and services



Accessors are proxies for IoT devices or services that execute in an "accessor host"

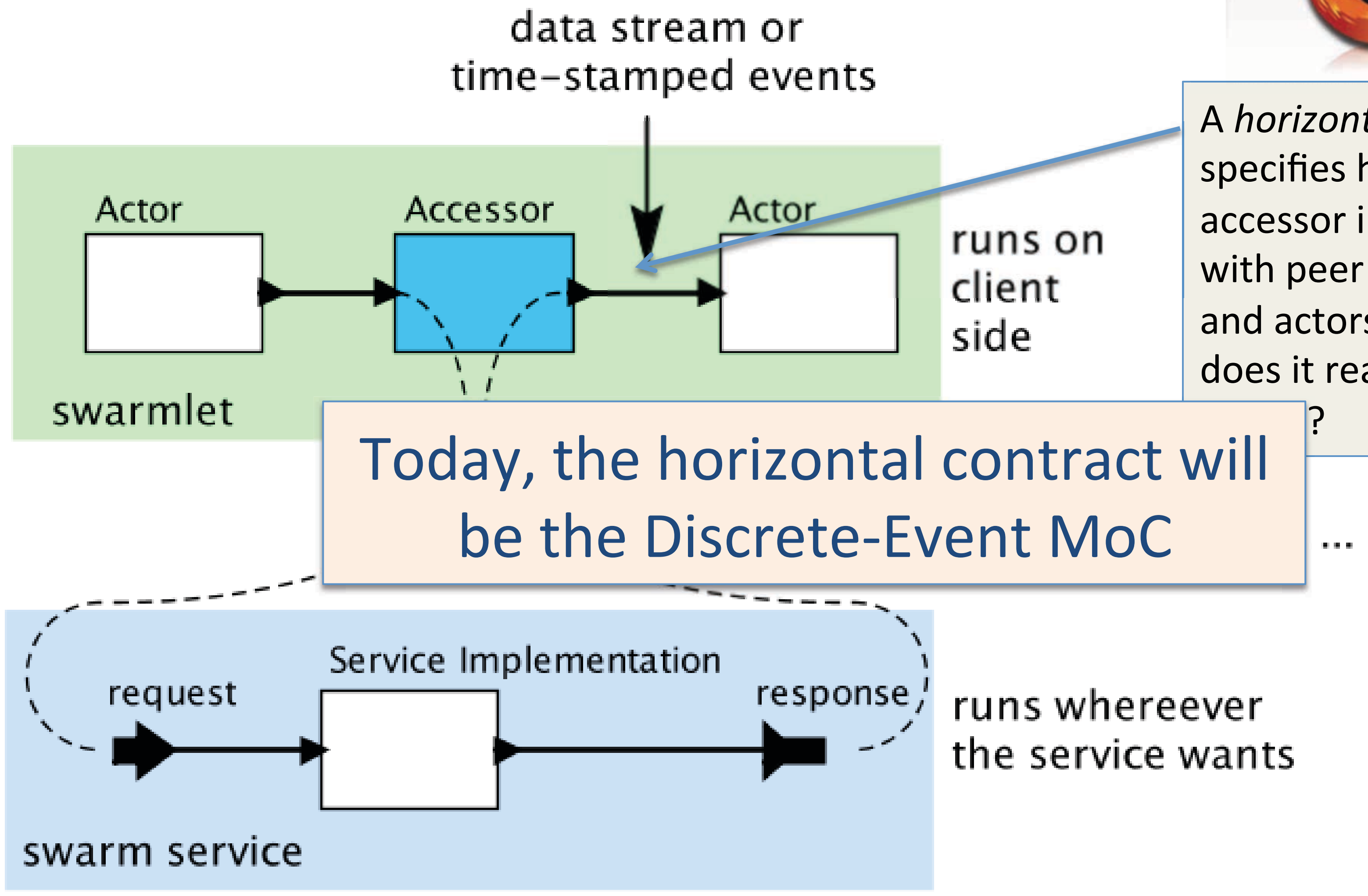
An *accessor host* is to IoT what a browser is to I



An **accessor host** instantiates and executes an **accessor**, which serves as a local proxy for a remote … or service

Today, the host will be Ptolemy II running on your laptop.

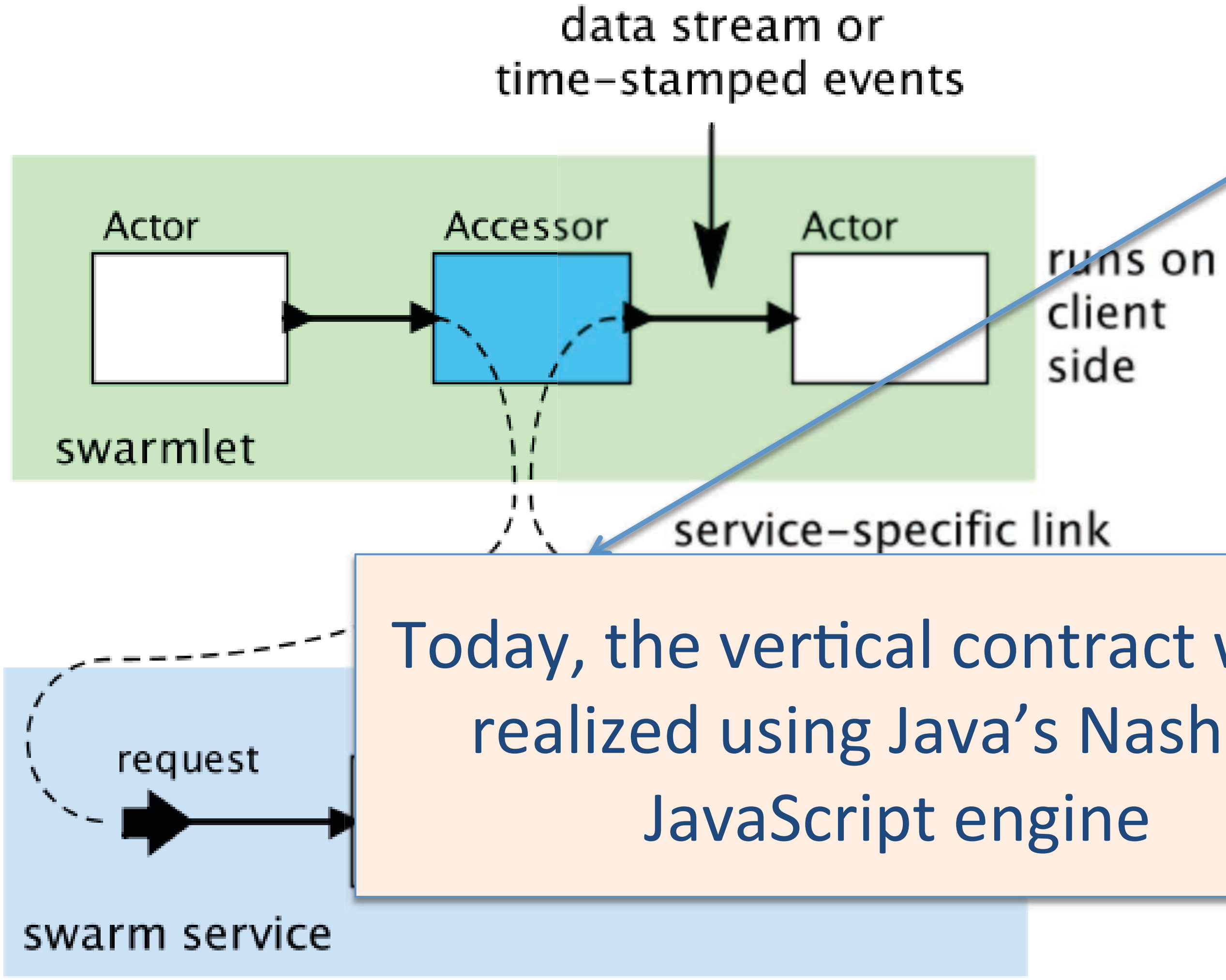
Horizontal Contracts



A *horizontal contact* specifies how the accessor interacts with peer accessors and actors. How does it react to ?

Today, the horizontal contract will be the Discrete-Event MoC

Vertical Contracts

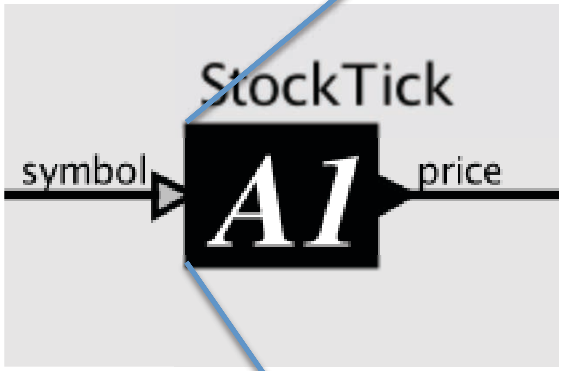


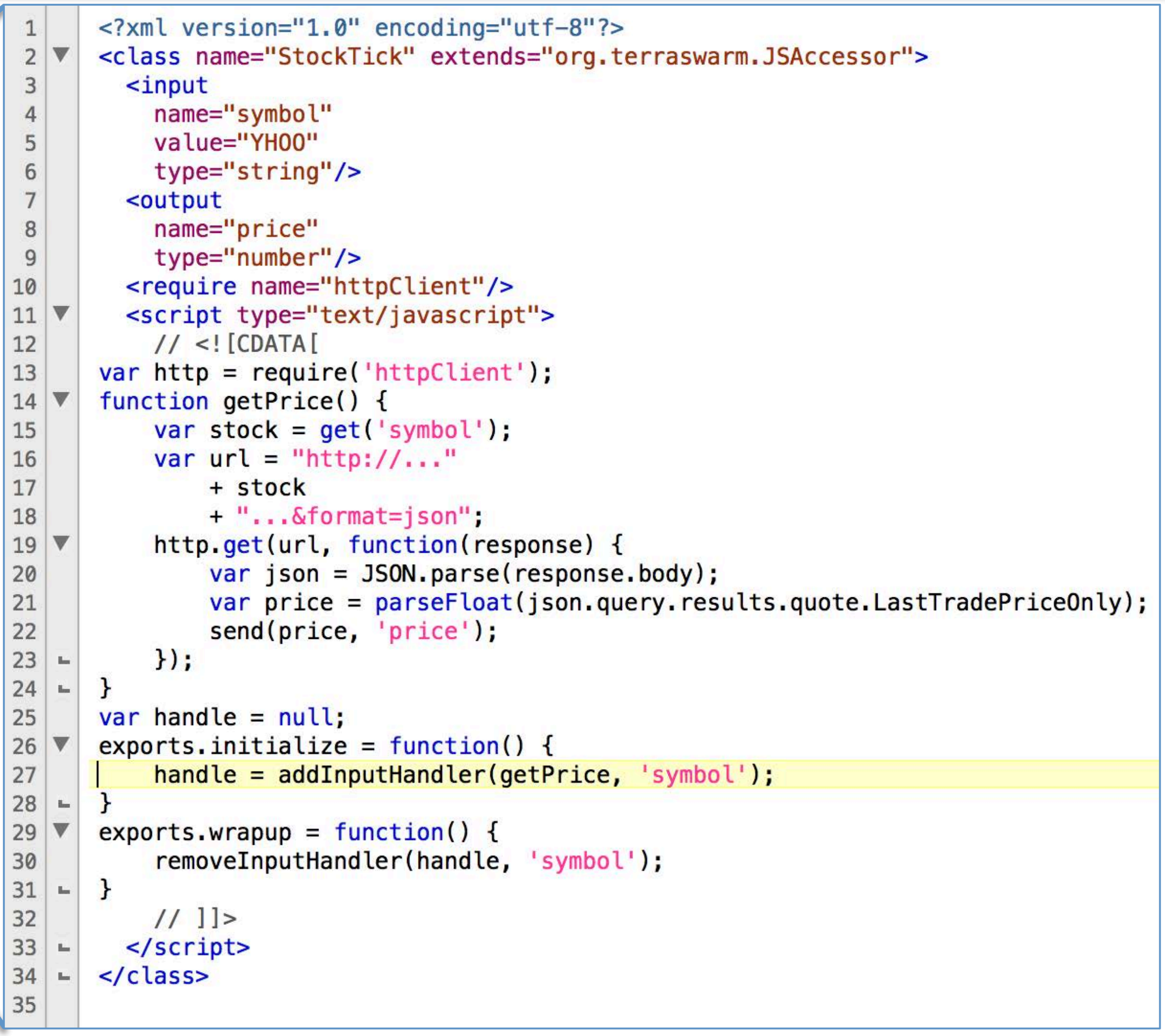
A *vertical contact* specifies how the accessor implements the service, including how it communicates with the service provider

…

Today, the vertical contract will be realized using Java's Nashorn JavaScript engine

The StockTick Accessor XML Syntax

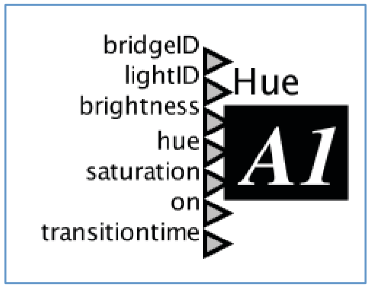




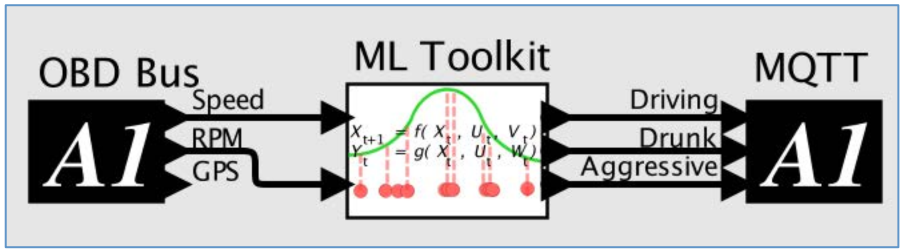
```
<?xml version="1.0" encoding="utf-8"?>
<class name="StockTick" extends="org.terraswarm.JSAccessor">
  <input
    name="symbol"
    value="YHOO"
    type="string"/>
  <output
    name="price"
    type="number"/>
  <require name="httpClient"/>
  <script type="text/javascript">
    // <![CDATA[
var http = require('httpClient');
function getPrice() {
    var stock = get('symbol');
    var url = "http://..."
        + stock
        + "...&format=json";
    http.get(url, function(response) {
        var json = JSON.parse(response.body);
        var price = parseFloat(json.query.results.quote.LastTradePriceOnly);
        send(price, 'price');
    });
}
var handle = null;
exports.initialize = function() {
    handle = addInputHandler(getPrice, 'symbol');
}
exports.wrapup = function() {
    removeInputHandler(handle, 'symbol');
}
    // ]]>
  </script>
</class>
```

Levels of Accessors

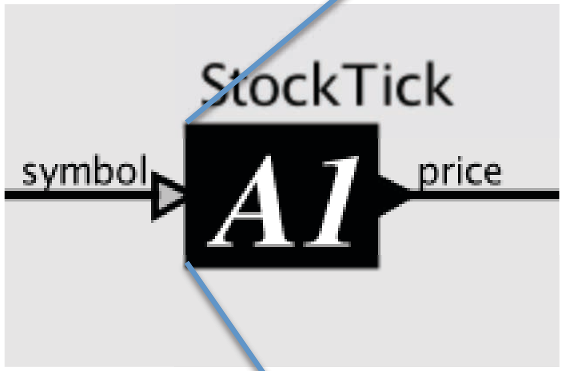
- Interface
  - Inputs, outputs, parameters
  - Data types, ontologies, protocols
- Component
  - Interface + Functionality
  - First prototype uses JavaScript
- Composition
  - Concurrent MoC
  - Distributed swarmlets

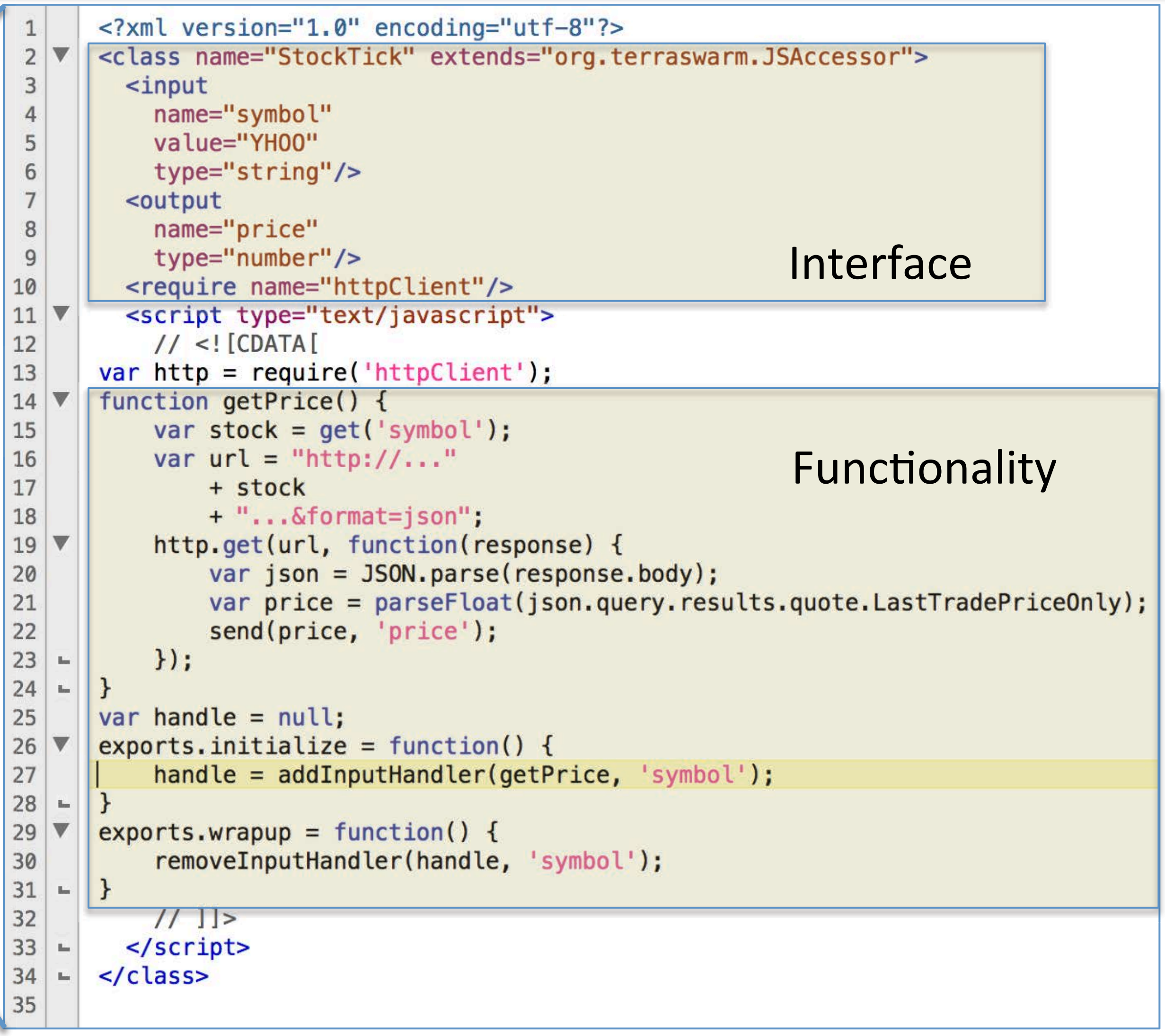


```
function fire() {
  var command = '{"on":false,';
  if (get(on) === "true") {
    command = '{"on":true,';
  } …
```



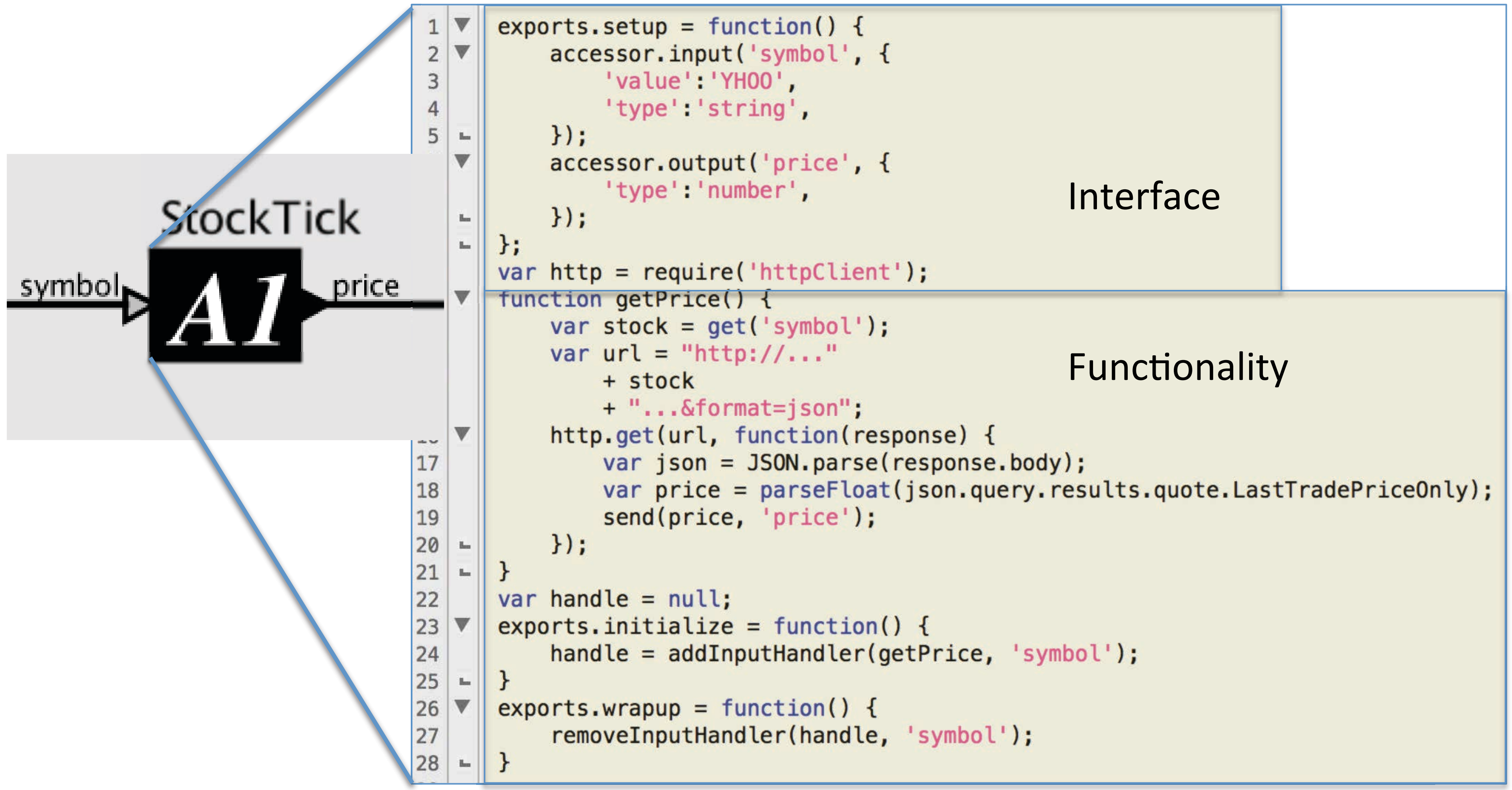
The StockTick Accessor XML Syntax





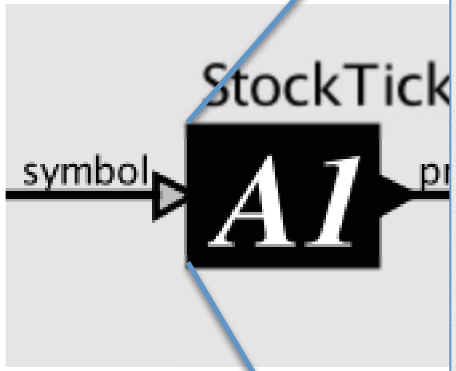
```
<?xml version="1.0" encoding="utf-8"?>
<class name="StockTick" extends="org.terraswarm.JSAccessor">
  <input
    name="symbol"
    value="YHOO"
    type="string"/>
  <output
    name="price"
    type="number"/>
  <require name="httpClient"/>
  <script type="text/javascript">
    // <![CDATA[
var http = require('httpClient');
function getPrice() {
    var stock = get('symbol');
    var url = "http://..."
        + stock
        + "...&format=json";
    http.get(url, function(response) {
        var json = JSON.parse(response.body);
        var price = parseFloat(json.query.results.quote.LastTradePriceOnly);
        send(price, 'price');
    });
}
var handle = null;
exports.initialize = function() {
    handle = addInputHandler(getPrice, 'symbol');
}
exports.wrapup = function() {
    removeInputHandler(handle, 'symbol');
}
    // ]]>
  </script>
</class>
```

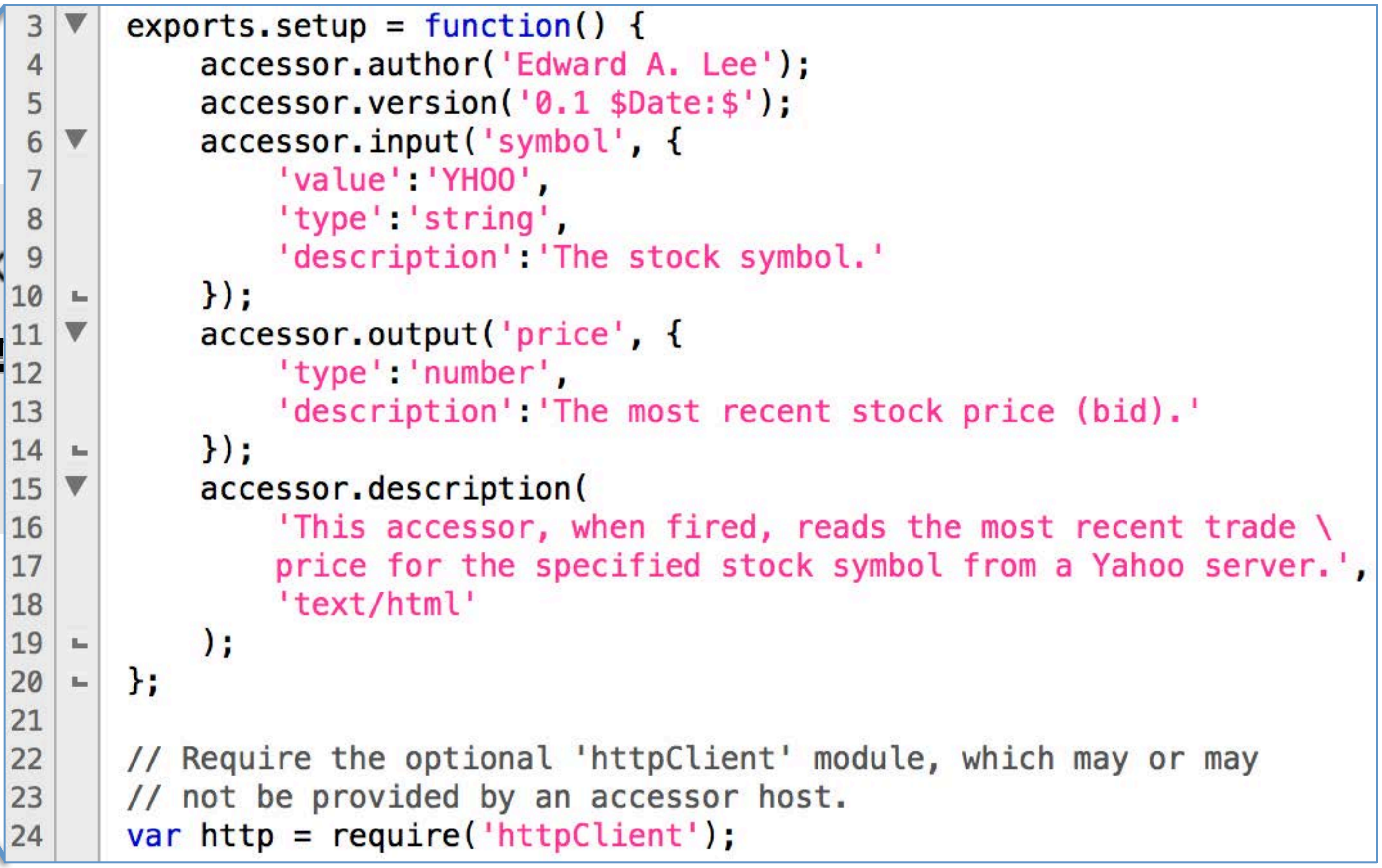
The StockTick Accessor JavaScript Syntax



```
exports.setup = function() {
    accessor.input('symbol', {
        'value':'YHOO',
        'type':'string',
    });
    accessor.output('price', {
        'type':'number',
    });
};
var http = require('httpClient');
function getPrice() {
    var stock = get('symbol');
    var url = "http://..."
        + stock
        + "...&format=json";
    http.get(url, function(response) {
        var json = JSON.parse(response.body);
        var price = parseFloat(json.query.results.quote.LastTradePriceOnly);
        send(price, 'price');
    });
}
var handle = null;
exports.initialize = function() {
    handle = addInputHandler(getPrice, 'symbol');
}
exports.wrapup = function() {
    removeInputHandler(handle, 'symbol');
}
```

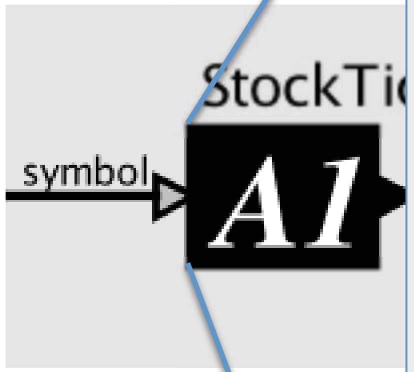
Interface Definition JavaScript Syntax

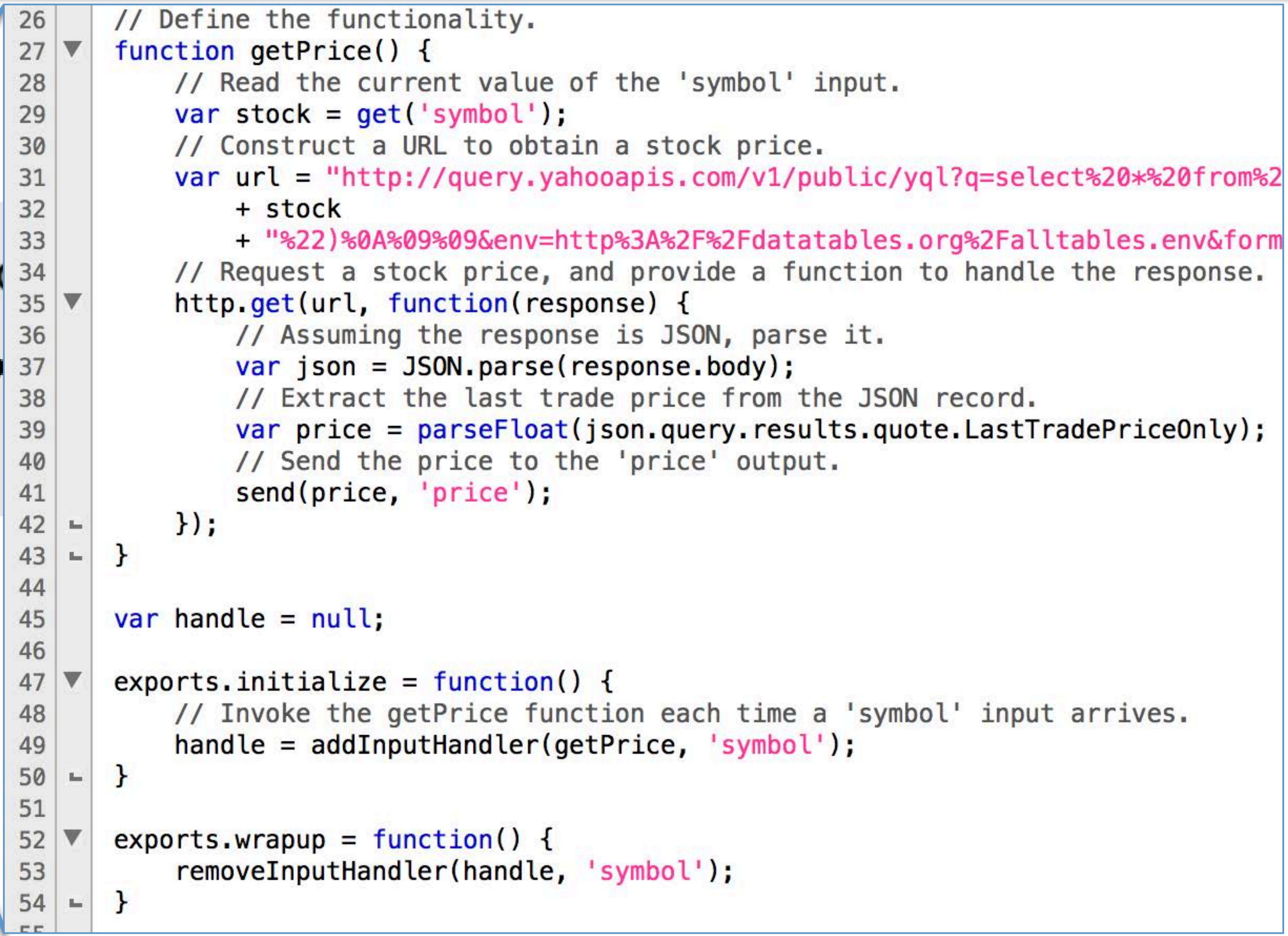




```
exports.setup = function() {
    accessor.author('Edward A. Lee');
    accessor.version('0.1 $Date:$');
    accessor.input('symbol', {
        'value':'YHOO',
        'type':'string',
        'description':'The stock symbol.'
    });
    accessor.output('price', {
        'type':'number',
        'description':'The most recent stock price (bid).'
    });
    accessor.description(
        'This accessor, when fired, reads the most recent trade \
        price for the specified stock symbol from a Yahoo server.',
        'text/html'
    );
};

// Require the optional 'httpClient' module, which may or may
// not be provided by an accessor host.
var http = require('httpClient');
```

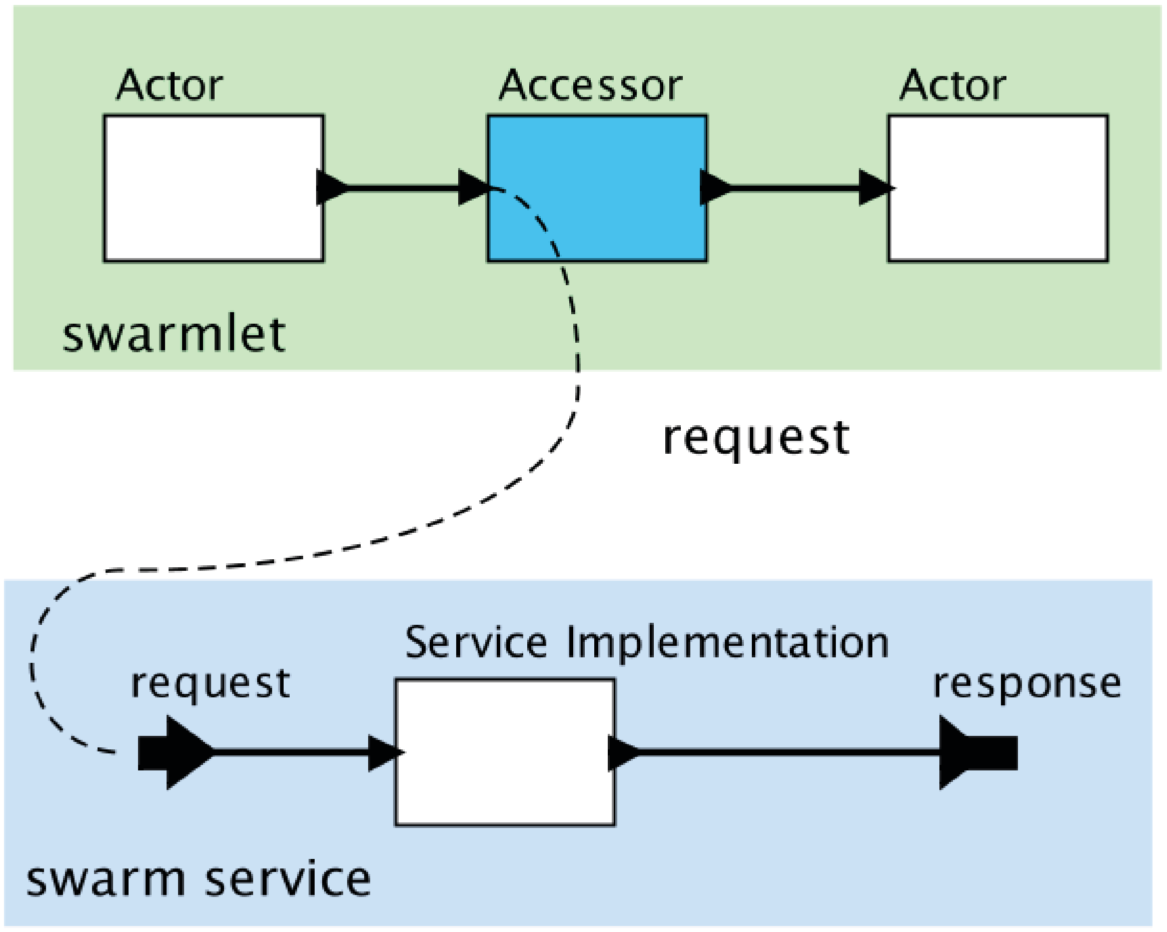
Functionality Definition JavaScript Syntax



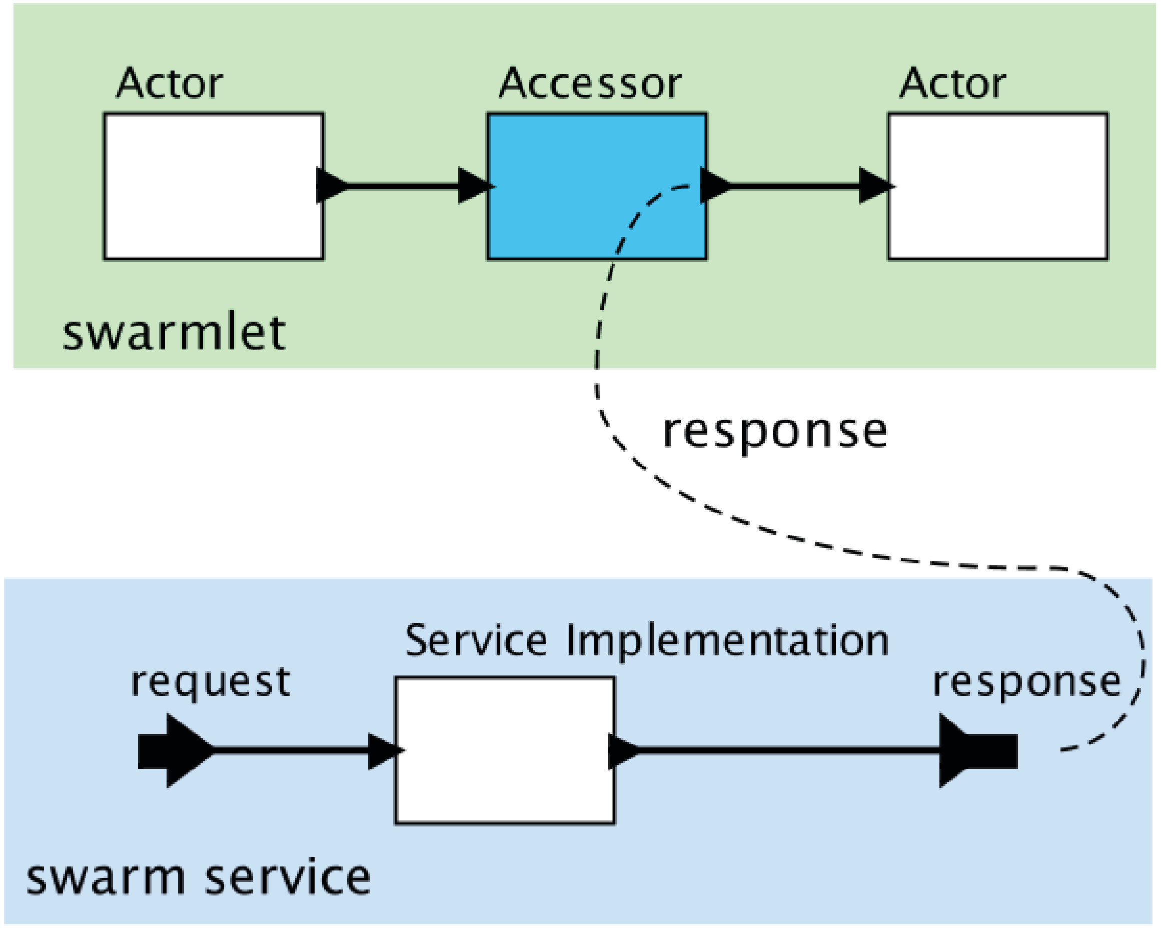


```
// Define the functionality.
function getPrice() {
    // Read the current value of the 'symbol' input.
    var stock = get('symbol');
    // Construct a URL to obtain a stock price.
    var url = "http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%2
        + stock
        + "%22)%0A%09%09&env=http%3A%2F%2Fdatatables.org%2Falltables.env&form
    // Request a stock price, and provide a function to handle the response.
    http.get(url, function(response) {
        // Assuming the response is JSON, parse it.
        var json = JSON.parse(response.body);
        // Extract the last trade price from the JSON record.
        var price = parseFloat(json.query.results.quote.LastTradePriceOnly);
        // Send the price to the 'price' output.
        send(price, 'price');
    });
}

var handle = null;

exports.initialize = function() {
    // Invoke the getPrice function each time a 'symbol' input arrives.
    handle = addInputHandler(getPrice, 'symbol');
}

exports.wrapup = function() {
    removeInputHandler(handle, 'symbol');
}
```

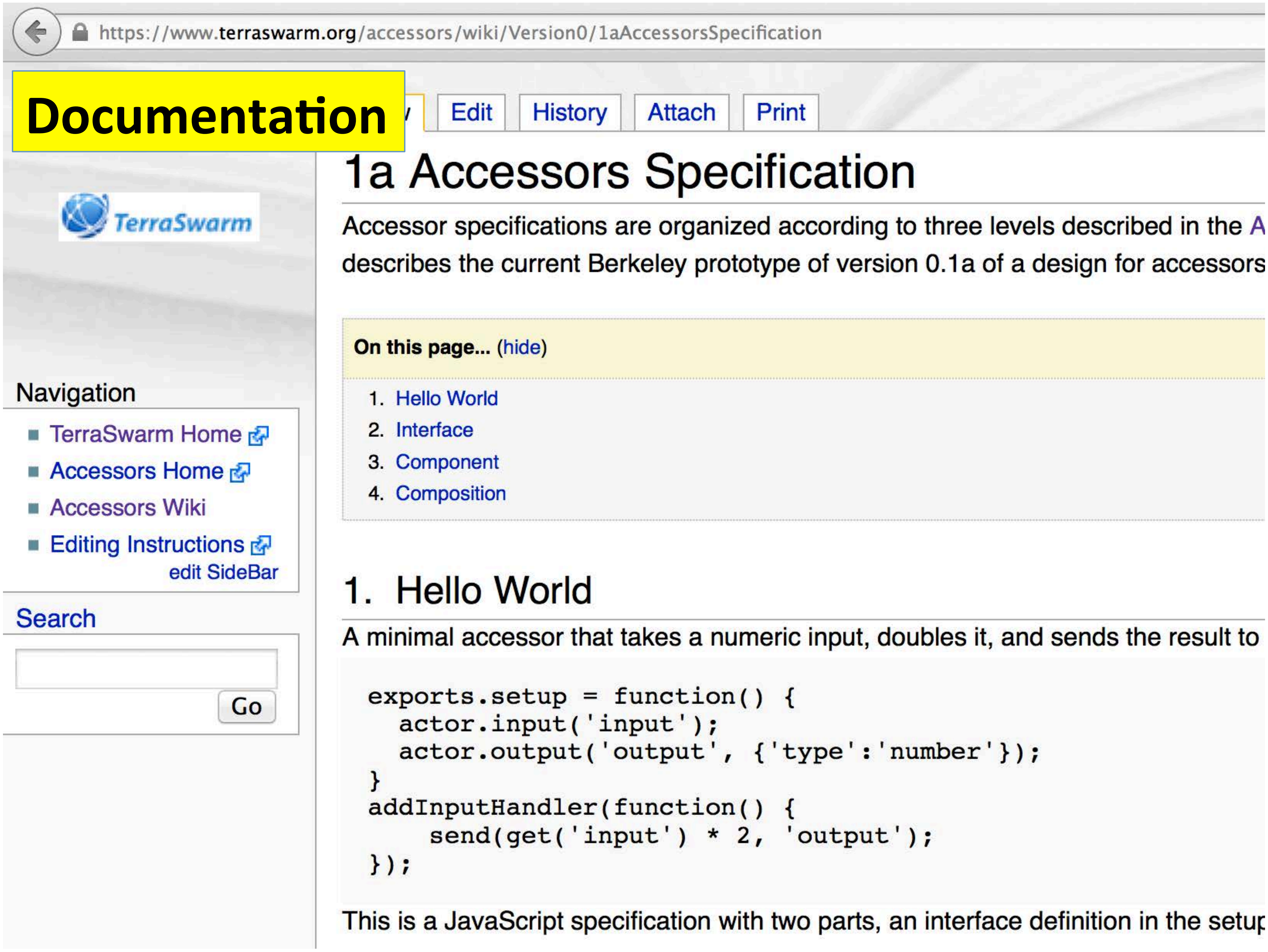
StockTick is an Asynchronous Accessor



StockTick is an Asynchronous Accessor



Documentation


https://www.terraswarm.org/accessors/wiki/Version0/1aAccessorsSpecification

Navigation

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

edit SideBar

Search

Go

1a Accessors Specification

Accessor specifications are organized according to three levels described in the A
describes the current Berkeley prototype of version 0.1a of a design for accessors

On this page... (hide)

1. Hello World

A minimal accessor that takes a numeric input, doubles it, and sends the result to

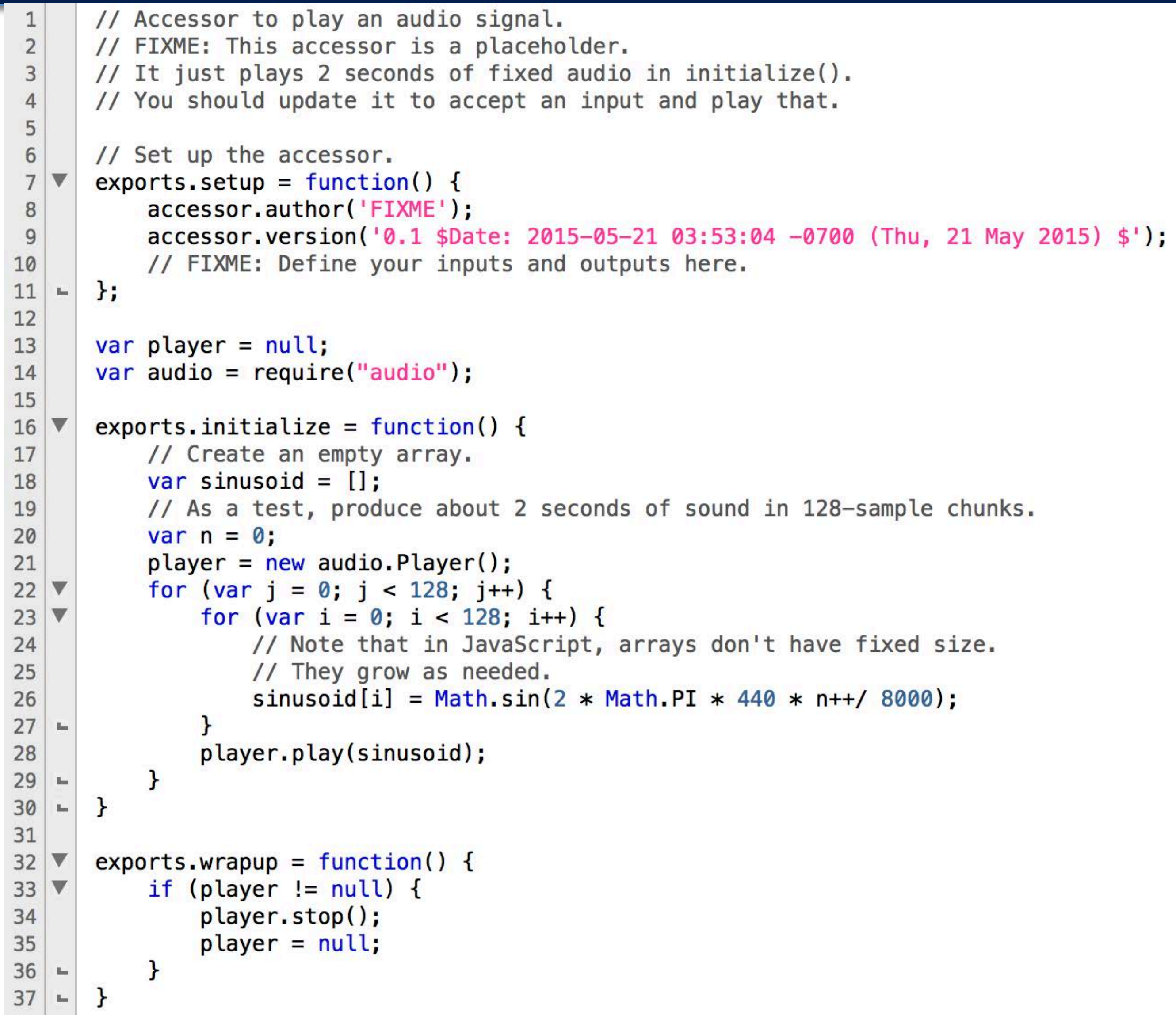
```
exports.setup = function() {
  actor.input('input');
  actor.output('output', {'type':'number'});
}
addInputHandler(function() {
    send(get('input') * 2, 'output');
});
```

This is a JavaScript specification with two parts, an interface definition in the setup

Common Exercise: Part 1

Incomplete accessor that has no inputs nor outputs and plays a fixed (annoying) sinusoidal sound.

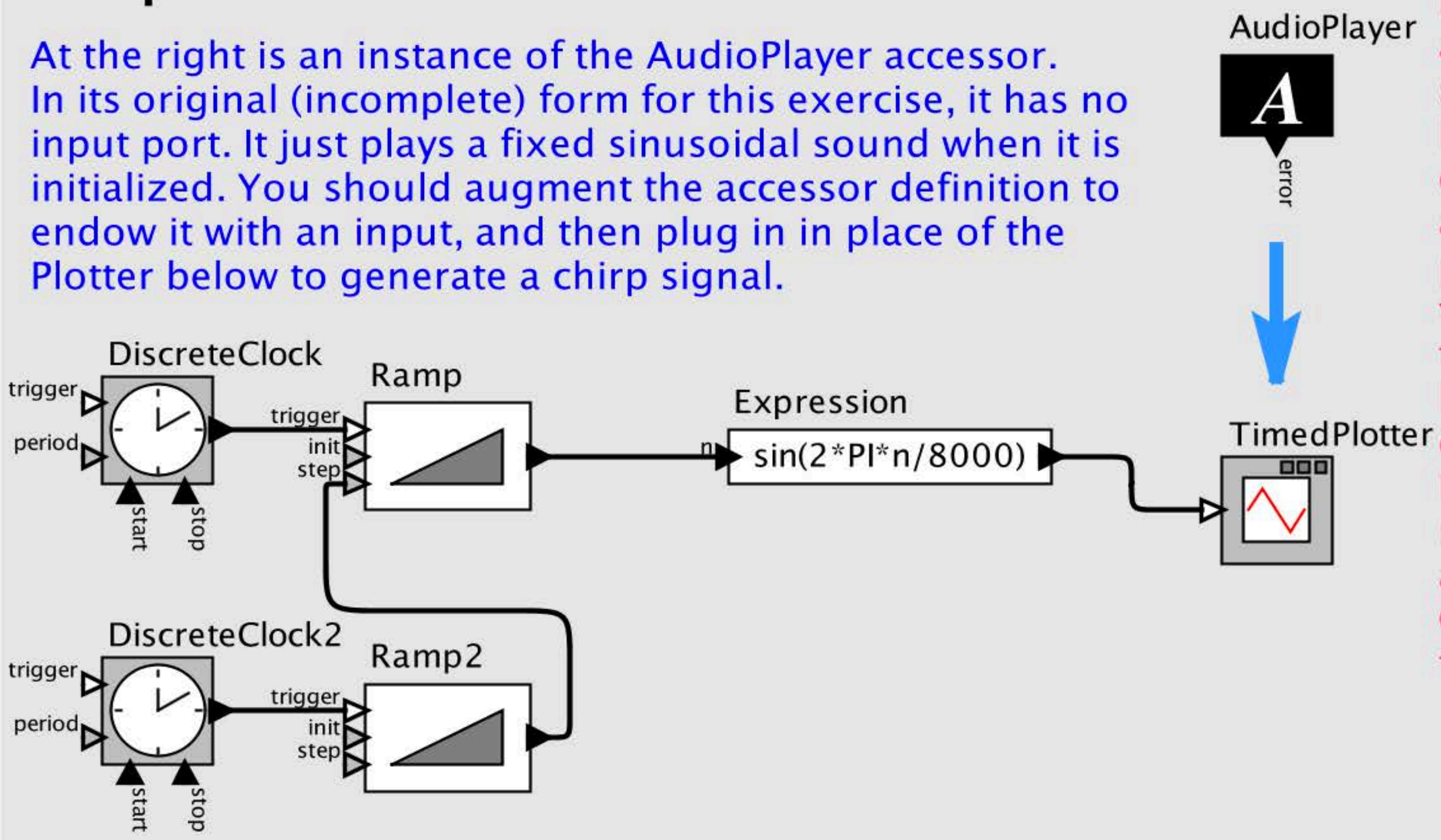
Your job: Make it better.



```javascript
// Accessor to play an audio signal.
// FIXME: This accessor is a placeholder.
// It just plays 2 seconds of fixed audio in initialize().
// You should update it to accept an input and play that.

// Set up the accessor.
exports.setup = function() {
    accessor.author('FIXME');
    accessor.version('0.1 $Date: 2015-05-21 03:53:04 -0700 (Thu, 21 May 2015) $');
    // FIXME: Define your inputs and outputs here.
};

var player = null;
var audio = require("audio");

exports.initialize = function() {
    // Create an empty array.
    var sinusoid = [];
    // As a test, produce about 2 seconds of sound in 128-sample chunks.
    var n = 0;
    player = new audio.Player();
    for (var j = 0; j < 128; j++) {
        for (var i = 0; i < 128; i++) {
            // Note that in JavaScript, arrays don't have fixed size.
            // They grow as needed.
            sinusoid[i] = Math.sin(2 * Math.PI * 440 * n++/ 8000);
        }
        player.play(sinusoid);
    }
}

exports.wrapup = function() {
    if (player != null) {
        player.stop();
        player = null;
    }
}
```

Template for Accessor Exercise

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.

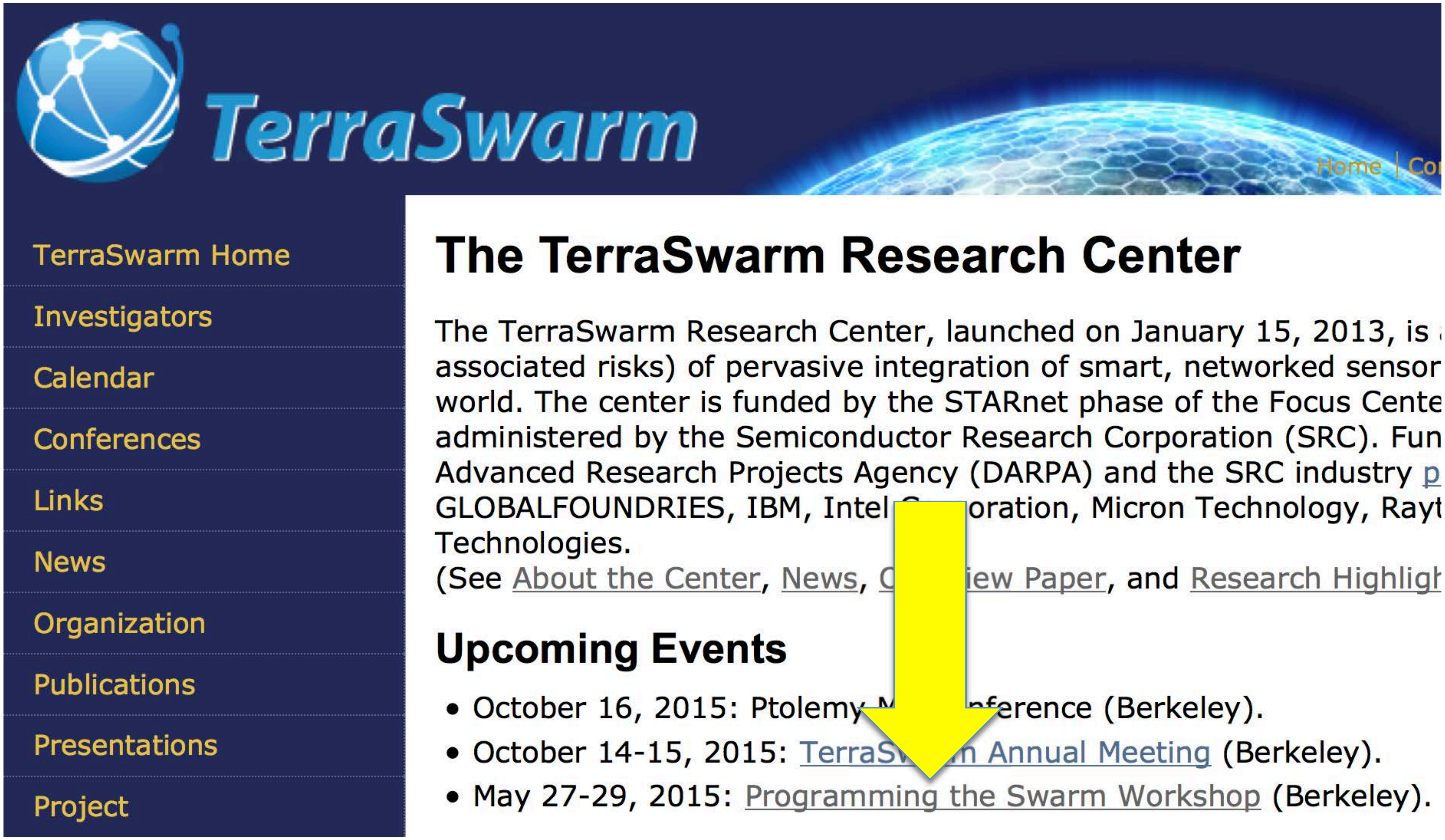


Download the accessor source, store on your local disk, change the accessorSource parameter of the accessor to point to your local copy, and click Reload. Now you can modify the accessor design. Click Reload for each update.

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.

Lost Yet?
http://terraswarm.org



TerraSwarm

Home | Co

TerraSwarm Home
Investigators
Calendar
Conferences
Links
News
Organization
Publications
Presentations
Project

The TerraSwarm Research Center

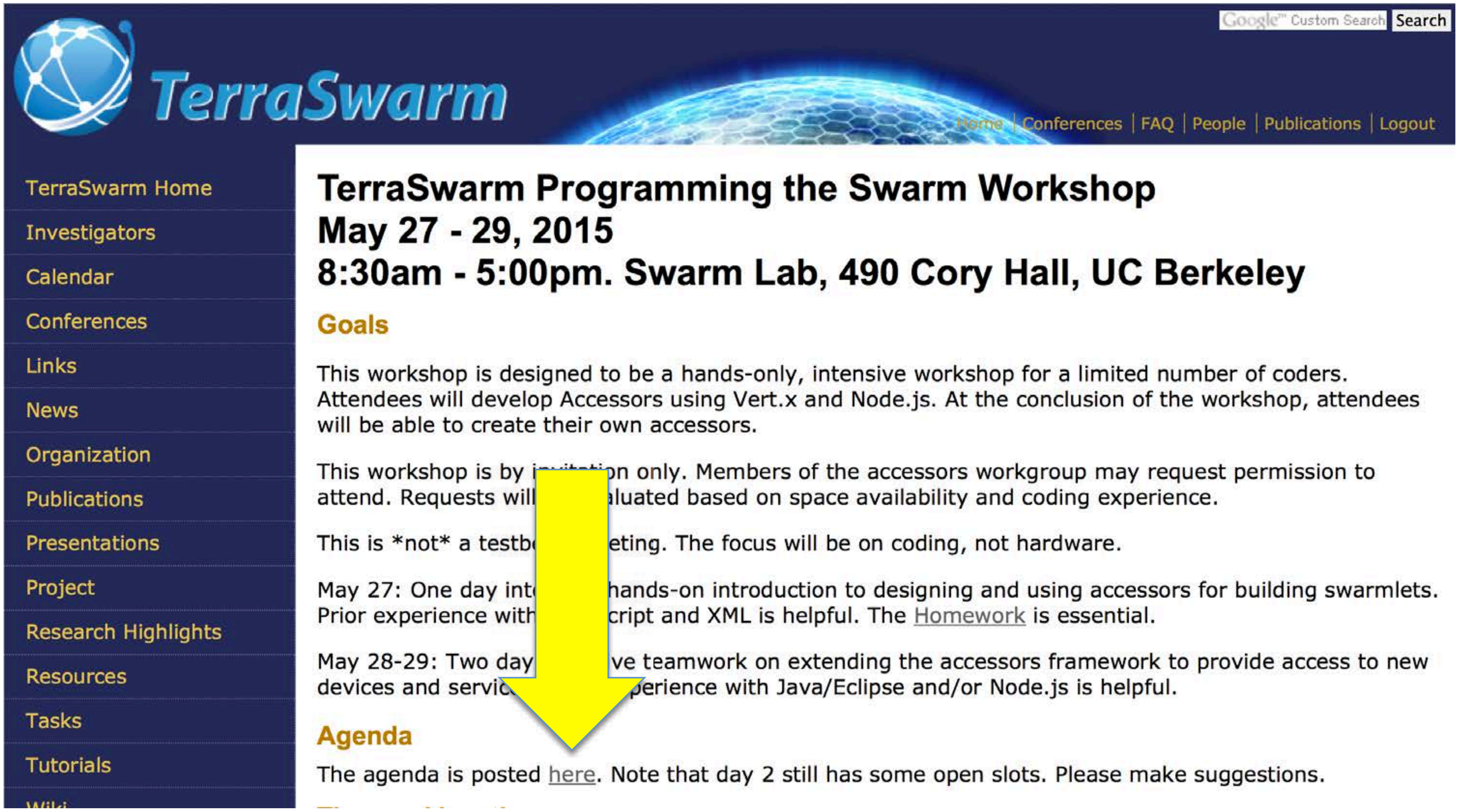
The TerraSwarm Research Center, launched on January 15, 2013, is
associated risks) of pervasive integration of smart, networked sensor
world. The center is funded by the STARnet phase of the Focus Cente
administered by the Semiconductor Research Corporation (SRC). Fun
Advanced Research Projects Agency (DARPA) and the SRC industry p
GLOBALFOUNDRIES, IBM, Intel C oration, Micron Technology, Rayt
Technologies.
(See About the Center, News, C iew Paper, and Research Highligh

Upcoming Events

- October 16, 2015: Ptolemy M nference (Berkeley).
- October 14-15, 2015: TerraSw n Annual Meeting (Berkeley).
- May 27-29, 2015: Programming the Swarm Workshop (Berkeley).

Lost Yet?

http://terraswarm.org/conferences/15/ProgrammingTheSwarmWorkshop



TerraSwarm Programming the Swarm Workshop
May 27 - 29, 2015
8:30am - 5:00pm. Swarm Lab, 490 Cory Hall, UC Berkeley

Goals

This workshop is designed to be a hands-only, intensive workshop for a limited number of coders. Attendees will develop Accessors using Vert.x and Node.js. At the conclusion of the workshop, attendees will be able to create their own accessors.

This workshop is by i[illegible]on only. Members of the accessors workgroup may request permission to attend. Requests will [illegible]aluated based on space availability and coding experience.

This is *not* a testb[illegible]eting. The focus will be on coding, not hardware.

May 27: One day int[illegible] hands-on introduction to designing and using accessors for building swarmlets. Prior experience with [illegible]cript and XML is helpful. The Homework is essential.

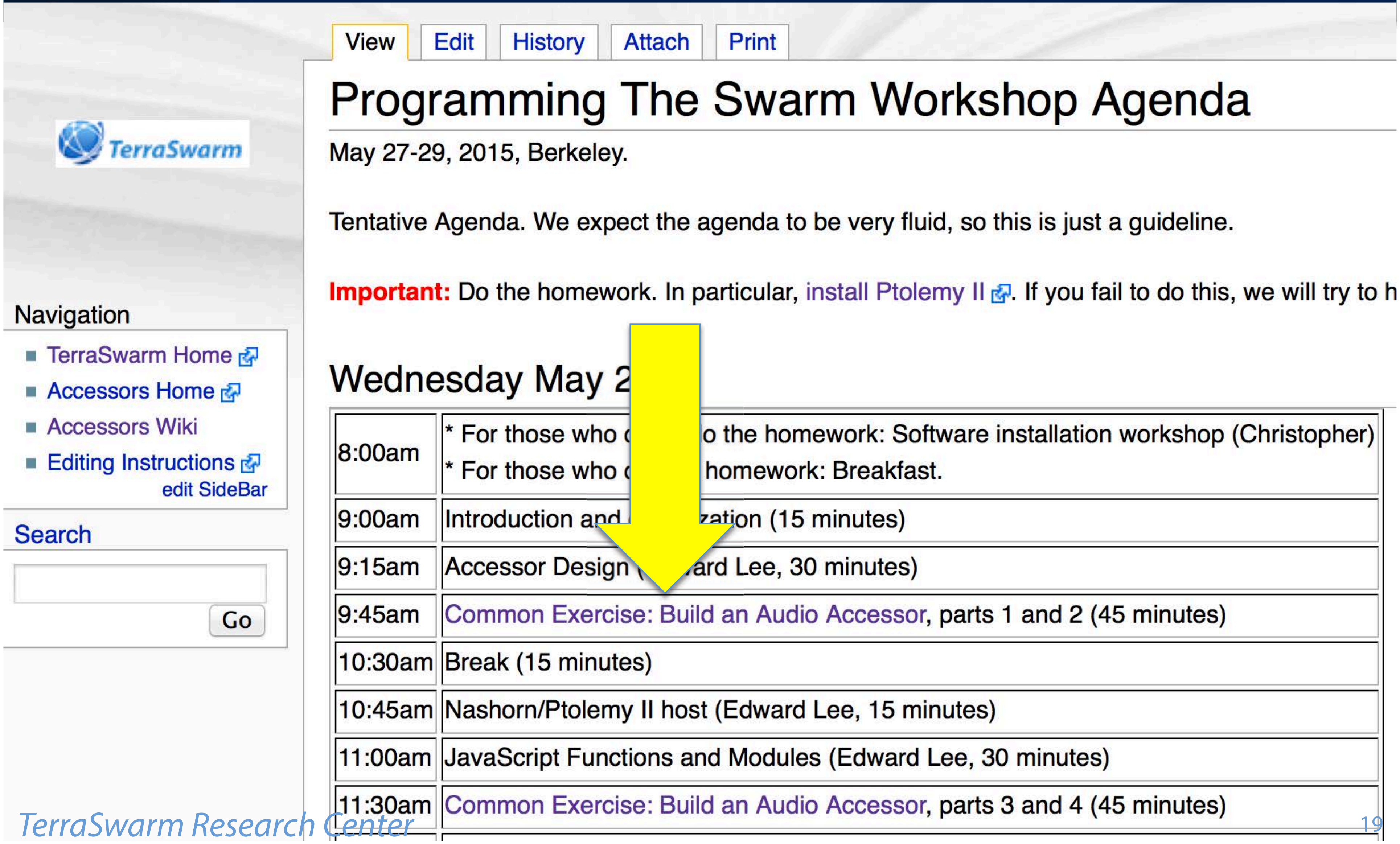
May 28-29: Two day[illegible]ve teamwork on extending the accessors framework to provide access to new devices and servic[illegible]perience with Java/Eclipse and/or Node.js is helpful.

Agenda

The agenda is posted here. Note that day 2 still has some open slots. Please make suggestions.

Lost Yet?

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



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.

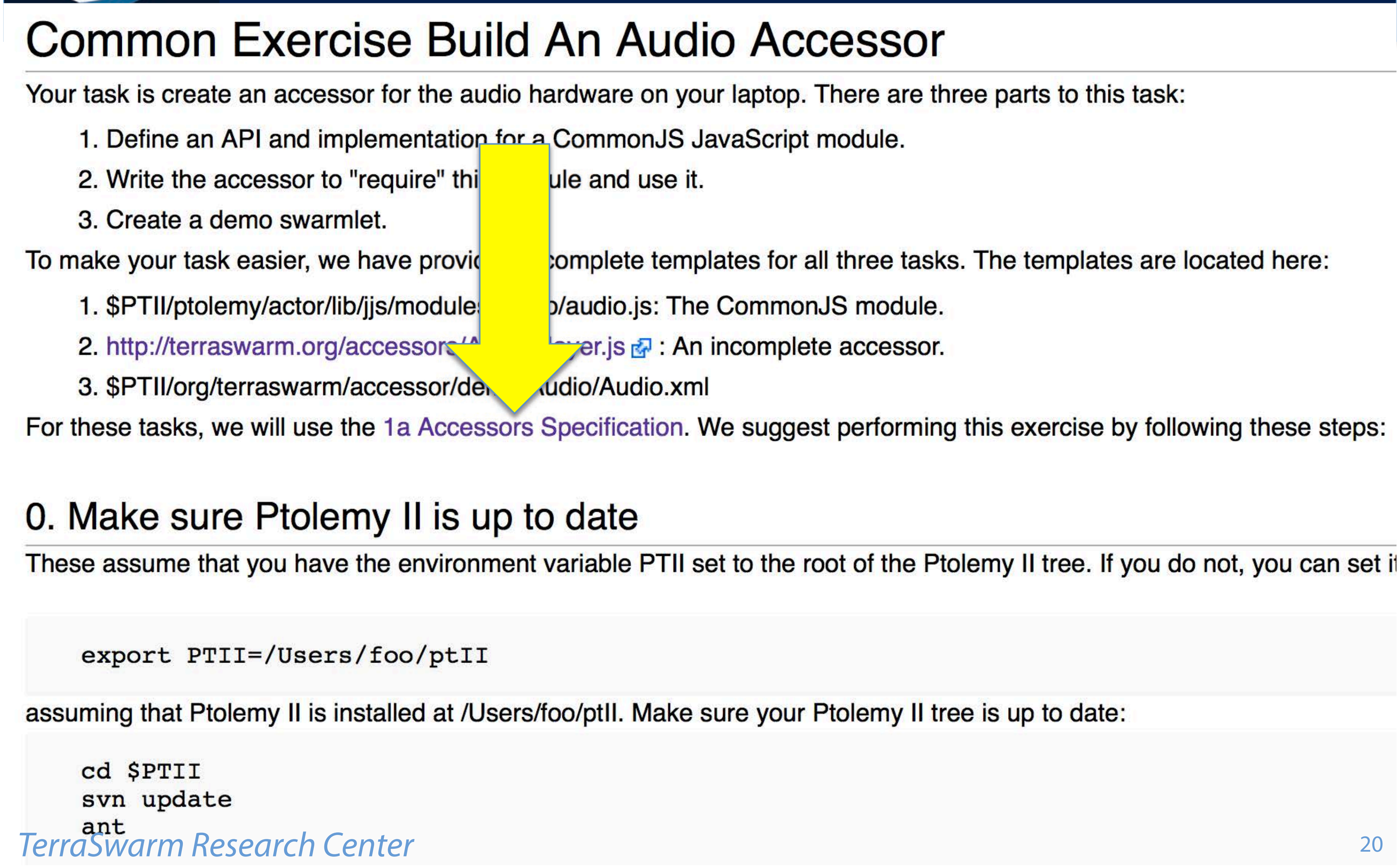
Important: Do the homework. In particular, install Ptolemy II. If you fail to do this, we will try to h

Wednesday May 2

8:00am	* For those who o the homework: Software installation workshop (Christopher) * For those who homework: Breakfast.
9:00am	Introduction and zation (15 minutes)
9:15am	Accessor Design (ard Lee, 30 minutes)
9:45am	Common Exercise: Build 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)

Lost Yet?

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



Common Exercise Build An Audio Accessor

Your task is create an accessor for the audio hardware on your laptop. There are three parts to this task:

1. Define an API and implementation for a CommonJS JavaScript module.
2. Write the accessor to "require" thi ule and use it.
3. Create a demo swarmlet.

To make your task easier, we have provic complete templates for all three tasks. The templates are located here:

1. $PTII/ptolemy/actor/lib/jjs/module /audio.js: The CommonJS module.
2. http://terraswarm.org/accessors er.js : An incomplete accessor.
3. $PTII/org/terraswarm/accessor/de udio/Audio.xml

For these tasks, we will use the 1a Accessors Specification. We suggest performing this exercise by following these steps:

0. Make sure Ptolemy II is up to date

These assume that you have the environment variable PTII set to the root of the Ptolemy II tree. If you do not, you can set i

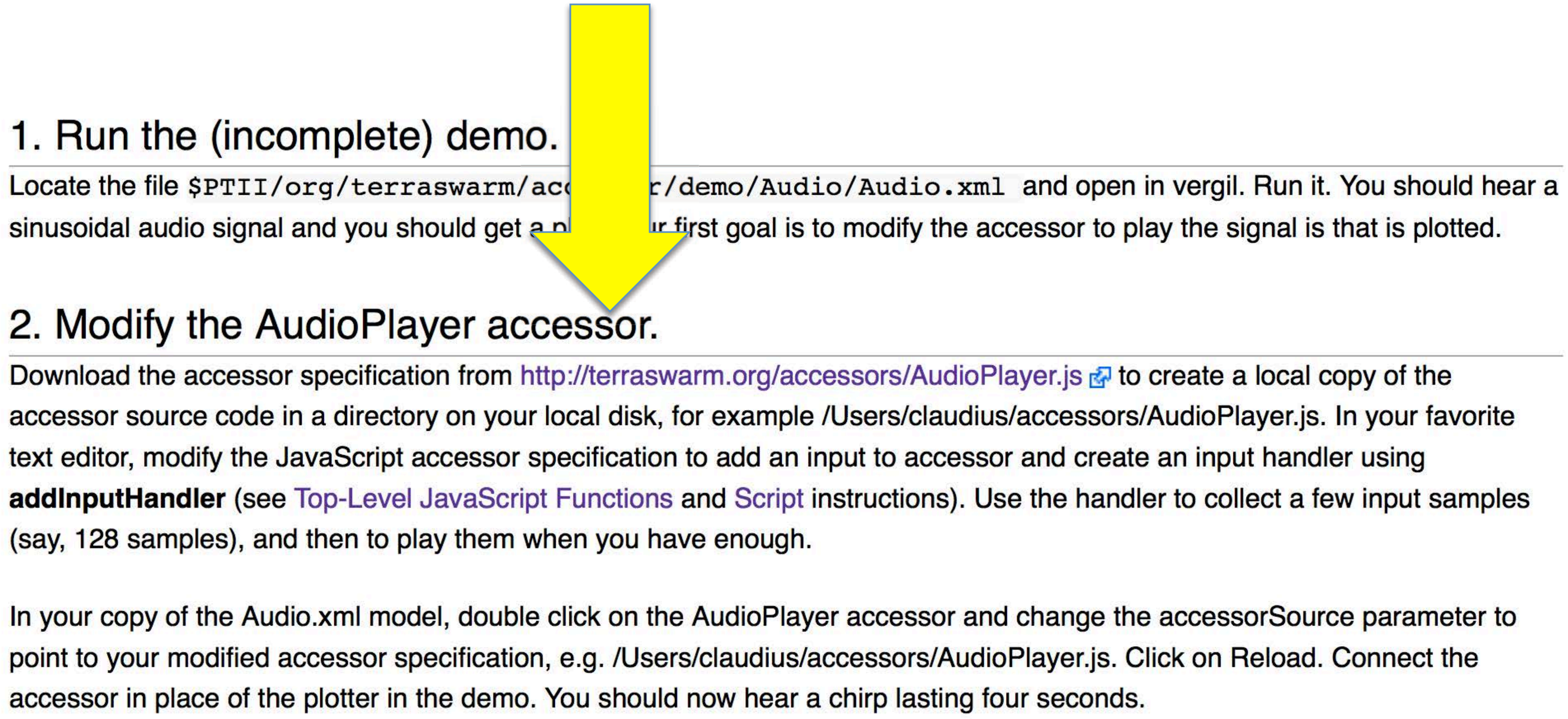
```
export PTII=/Users/foo/ptII
```

assuming that Ptolemy II is installed at /Users/foo/ptII. Make sure your Ptolemy II tree is up to date:

```
cd $PTII
svn update
ant
```

Lost Yet?

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



1. Run the (incomplete) demo.

Locate the file `$PTII/org/terraswarm/acc` `r/demo/Audio/Audio.xml` and open in vergil. Run it. You should hear a sinusoidal audio signal and you should get a p r first goal is to modify the accessor to play the signal is that is plotted.

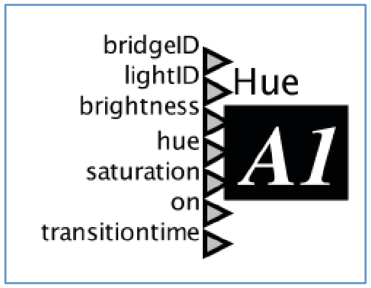
2. Modify the AudioPlayer accessor.

Download the accessor specification from http://terraswarm.org/accessors/AudioPlayer.js to create a local copy of the accessor source code in a directory on your local disk, for example /Users/claudius/accessors/AudioPlayer.js. In your favorite text editor, modify the JavaScript accessor specification to add an input to accessor and create an input handler using **addInputHandler** (see Top-Level JavaScript Functions and Script instructions). Use the handler to collect a few input samples (say, 128 samples), and then to play them when you have enough.

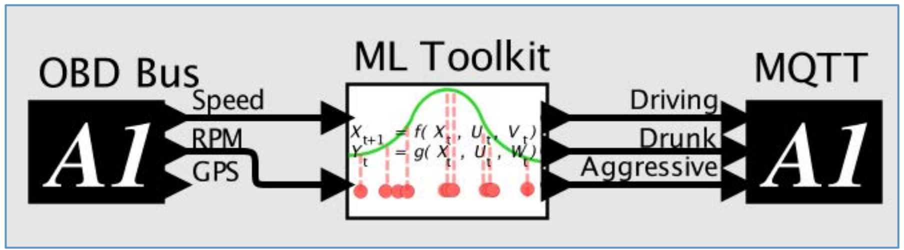
In your copy of the Audio.xml model, double click on the AudioPlayer accessor and change the accessorSource parameter to point to your modified accessor specification, e.g. /Users/claudius/accessors/AudioPlayer.js. Click on Reload. Connect the accessor in place of the plotter in the demo. You should now hear a chirp lasting four seconds.

Big Questions Remain (Try to not get distracted today)

- Interface
  - Subtyping?
  - Ontologies?
  - Contracts?
  - Discovery?
- Component
  - Languages?
  - Libraries?
  - Sandboxing?
  - Authentication?
  - Error handling?
- Composition
  - What MoCs?
  - Callbacks vs. actors?
  - Time stamping?
  - Always live swarmlets?



```
function fire() {
  var command = '{"on":false,';
  if (get(on) === "true") {
    command = '{"on":true,';
  } …
```


OBD Bus
A1
Speed
RPM
GPS
ML Toolkit
Driving
Drunk
Aggressive
MQTT
A1