CapeCode Tutorial

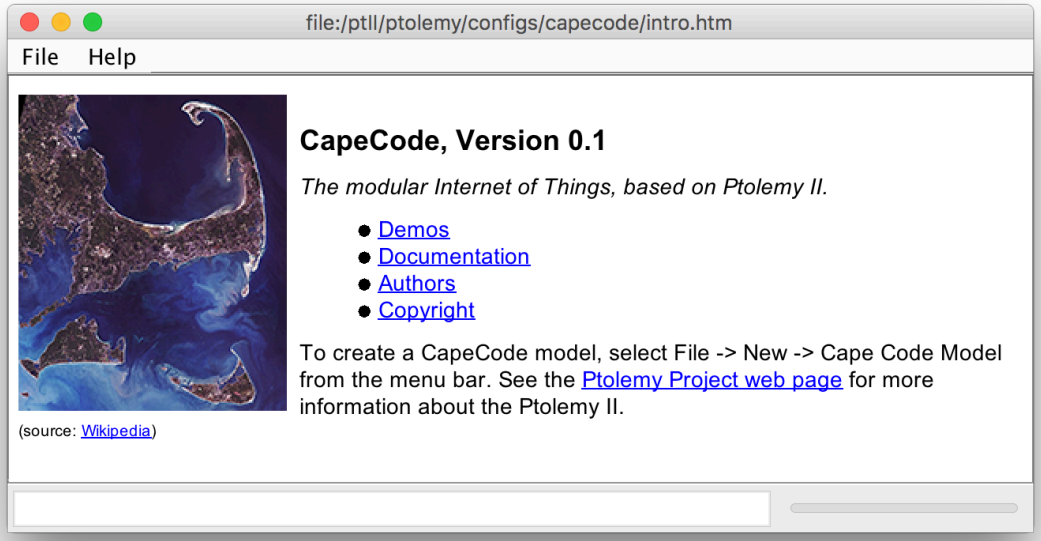
By Christopher Brooks and Edward A. Lee
Version 1.0, Dec. 5, 2017

Cape Code is interactive graphical editor for swarmlets based on Ptolemy II. If you have installed Ptolemy II (the current development version, not a release), then you can invoke Cape Code on the command line as follows:

```
> $PTII/bin/capecode
```

For convenience, you may want to add $PTII/bin to your PATH environment variable, in which case, you will not need to type $PTII/bin.

You will see the welcome window:

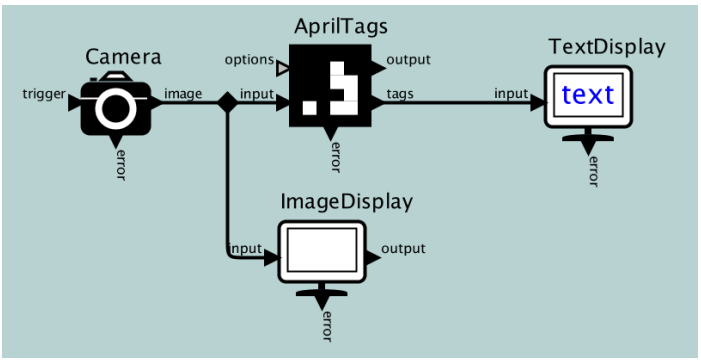


Cape Code is integrated with the TerraSwarm accessor library. The current contents of the library appears in the palette of available actors, as shown on the right below, and can be dragged into a model and connected with other actors or accessors in the library.

Cape Code includes a number of demos. Browse those by following the demos link in the welcome window shown above. The graphical editor is vergil, which is part of Ptolemy II. For a tutorial on Ptolemy II, see the Ptolemy Book (a free download).

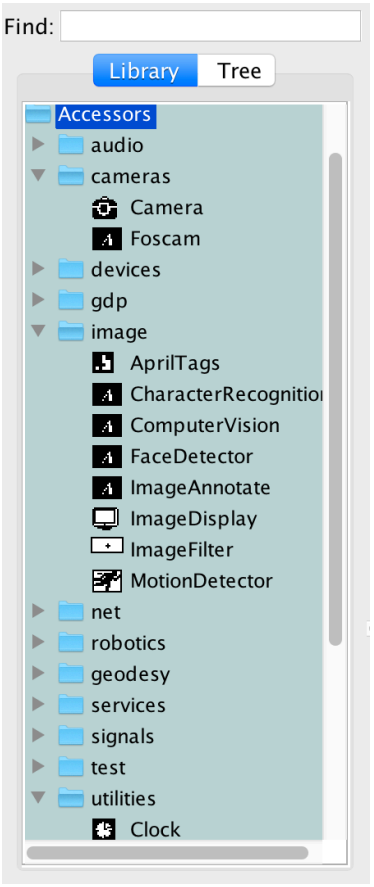
The files for this tutorial are located in Ptolemy II tree at $PTII/ptolemy/demo/CapeCode. Solutions to the exercises below are also in that directory, but we advise not looking at them until you get stuck.

1. Your first exercise is to build a model that uses your laptop's camera (via an accessor) to find AprilTags in the field of view and display their ID and location. To do this, construct a model that looks like this:



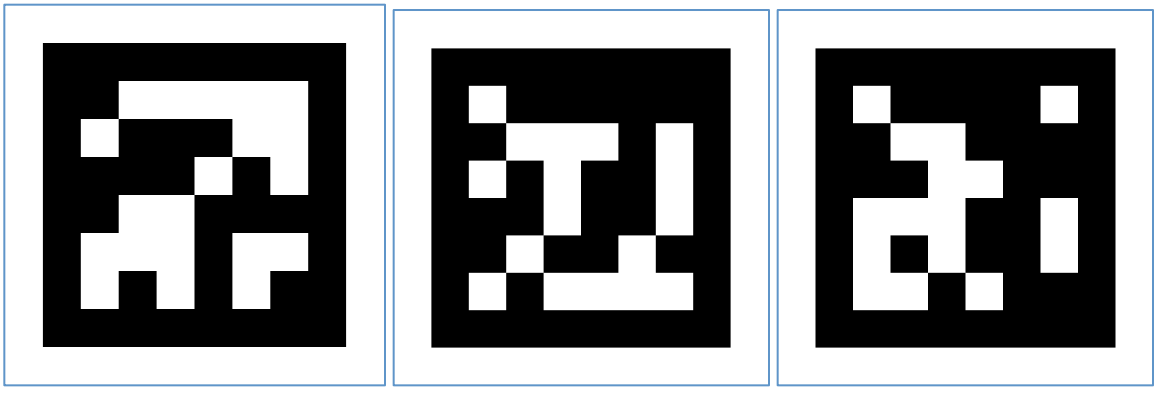
Each icon represents either an accessor or a Ptolemy actor. You can find them here in the CapeCode library:

Camera	Accessors → cameras
AprilTags	Accessors → image
TextDisplay	Accessors → utilities



A snapshot of the library is shown on the right. Drag in the relevant icons and connect them. (**Hint**: control-click or command-click will create a *relation*, the small black diamond above, which is used to fork a signal to multiple destinations.)

You can test your model using the following April tags, with IDs 19, 20, and 21:

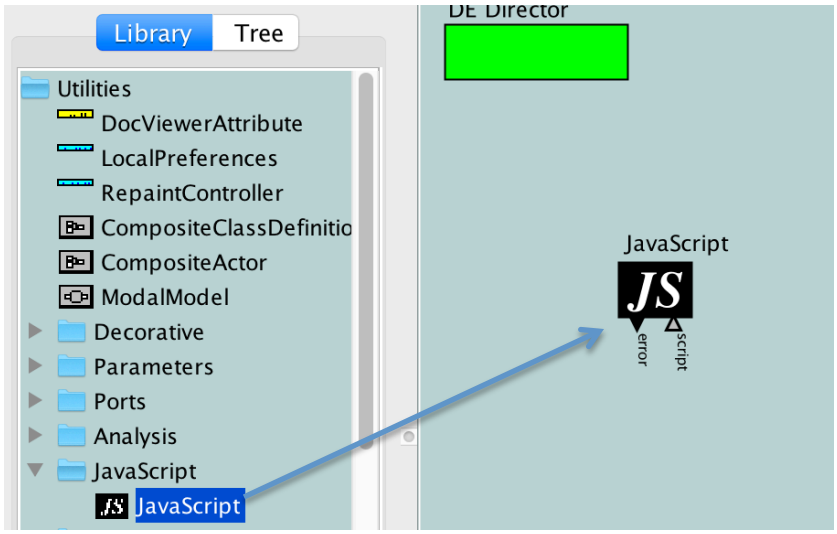


We suggest you reduce the *maxFrameRate* on the Camera to, say, 5. Double click on it to edit its parameters.

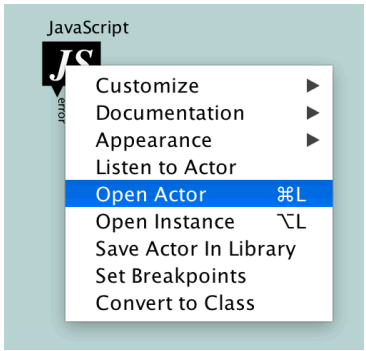
A solution to this exercise can be found in $PTII/ptolemy/demo/CapeCode/CapeCode1.xml.

2. Your second task is to augment the above model with a script that processes the tags from the AprilTags accessor and outputs a tag ID (a number) each time it sees a tag that was not present in the previous frame. In other words, you want one output event each time a *new* tag is observed by the camera.

   To realize this script, find the **JavaScript** actor in the **Utilities** library in CapeCode and drag an instance of that actor into your model as follows:



   Then open the actor with right- or control-click as shown below:

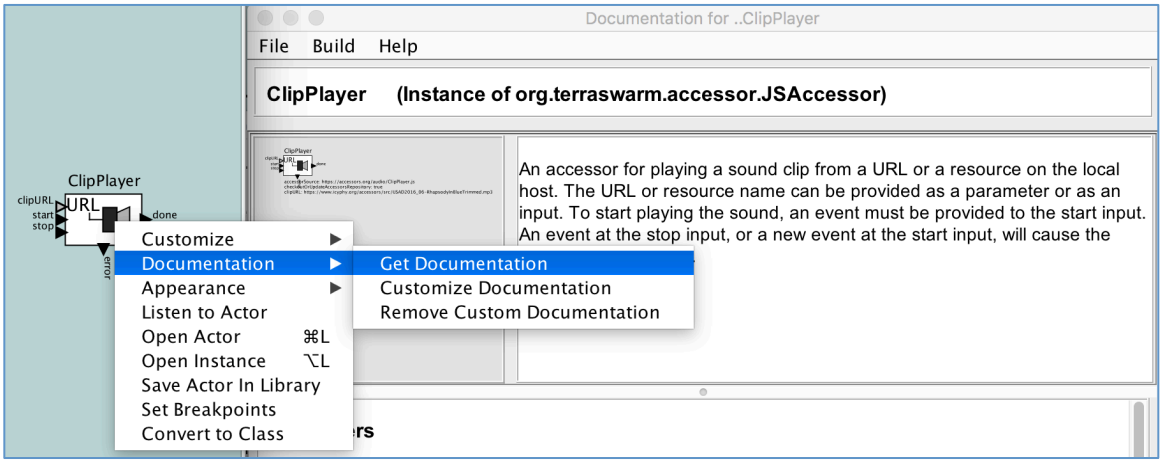


   Using the accompanying accessor tutorial document, enter a script into the resulting text box that defines an input port to receive each frame from the camera, an input port to receive each array of April tags from the **AprilTags** accessor, and an output port to produce the ID of each newly identified tag. You will want to add an input handler that is triggered each time a new image frame arrives.

   If you have never programmed in JavaScript before, feel free to study the solution found in $PTII/ptolemy/demo/CapeCode/CapeCode2.xml.

3. Your third task is to make a distinct sound each time your camera spots a new April tag. Specifically, you can use the **ClipPlayer** accessor, which has an input port that accepts a resource name or a URL that refers to a sound file. When it receives an event on its **start** input port, it will play the sound.

   Hint: Right- or control-click on an accessor or actor to get its documentation, as shown here:



   There are three sound files available to you using the following resource names:

```
$CLASSPATH/ptolemy/demo/CapeCode/Sounds/ring.wav
$CLASSPATH/ptolemy/demo/CapeCode/Sounds/bell.wav
$CLASSPATH/ptolemy/demo/CapeCode/Sounds/train.wav
```

   Your task is to play these three sounds in response to tags with IDs 19, 20, and 21, shown in the first exercise.

   To accomplish this task, we suggest first adding another **JavaScript** actor with a custom script that translates the IDs 19, 20, and 21 into the above resource names, then feeding the resulting resource names to the **clipURL** and **start** inputs of the **ClipPlayer.**

   A solution to this exercise can be found in $PTII/ptolemy/demo/CapeCode/CapeCode3.xml.

4. Your final task is to design an accessor for a remote service that produces the sound on another machine. If you are following this tutorial in a classroom setting, your instructor may have already set up the server on a machine running locally. Ask the instructor for the IP address and port of the server. Otherwise, you can start the server on localhost:8089 by opening the following model in CapeCode and running it:

   ```
   $PTII/ptolemy/demo/CapeCode/WebSoundServer.xml
   ```

   The server provides a RESTful interface, meaning that you can access the service using HTTP. An ordinary browser can be used to access the service. If you are running the above sound server on your local machine, then point your browser to the following URL to cause the service to produce a sound:

   ```
   http://localhost:8089/train
   ```

   The sounds provided are "train", "bell", and "ring", just like the previous exercise.

   Your accessor for this service should use the **http-client** module, which is described in the accompanying accessor tutorial document and also on this web page:

   https://www.icyphy.org/accessors/wiki/Version1/Http-client

   You will want your accessor to have a parameter specifying the IP address and port of the service, e.g. "`localhost:8089`". Alternatively, you could have two parameters, one for the host IP address or name and the other for the port.

   A solution to this exercise can be found in $PTII/ptolemy/demo/CapeCode/CapeCode4.xml.