

Build an application with CalvinGUI

- This guide presumes that at least part 1 and 2 of the scripts in the example has been successfully executed.

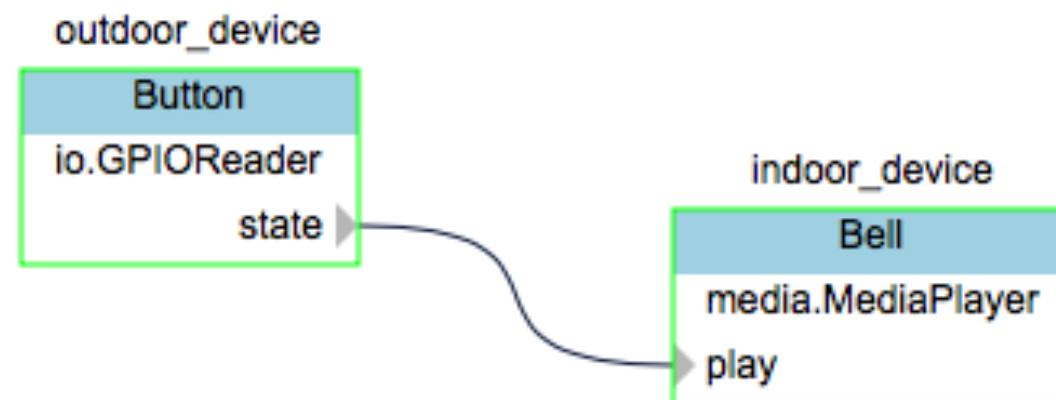
- Start three runtimes, one on the computer and one on each raspberry pi. Use the start scripts in the example:
 - `./start-computer.sh 127.0.0.1`
 - `./start-outdoor.sh <ip address>`
 - `./start-indoor.sh <ip address>`
- Goto the address: **http://94.246.117.105** in a web browser.
- Choose “**I have Calvin running on this machine**” and the address: **127.0.0.1:5001**
- Press **Start**

- ☐ Try Calvin in cloud sandbox
- ☒ I have Calvin running on this machine
- ☐ I have Calvin running on another machine on this network

Address:

Door Bell

- The following example shows how to build an application for a door bell in the Calvin GUI.



- Pick the actor `GPIOReader` found in category **io**, drag and drop it in the middle frame of the browser window.
- In the lower center frame, the arguments for initiating the actor are visible. Any red field indicates that an argument needs to be set to initiate the actor.

The screenshot displays a web-based actor editor interface. On the left, a sidebar lists categories: misc, net, sensor, std, test, text, time, and web. The main workspace shows an actor named `gpioreader1` of type `io.GPIOReader` with a `state` output. Below the workspace, a configuration panel for `io.GPIOReader` is visible, showing the actor's description, outputs, and requirements. The arguments table indicates that the `edge`, `gpio_pin`, and `pull` arguments need to be set, as their value fields are highlighted in red.

io.GPIOReader
Edge triggered GPIO state reader for pin <pin>

Outputs:
state : 0/1 when low/high edge detected

Requirements:
calvinsys.io.gpiohandler

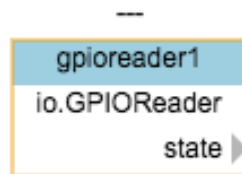
Argument	Value
edge	
gpio_pin	
pull	

io.GPIOReader
Edge triggered GPIO state reader for pin <pin>

Outputs:
state : 0/1 when low/high edge detected


Requirements:
calvinsys.io.gpiohandler

- Initiate the actor with the following values:
- edge=“**b**”
- gpio_pin=**23**
- pull=“**d**”
- It is good practice to name the actors in the application according to their purpose in the application. Change the name of GPIOReader to Button



Name: <input type="text" value="gpioreader1"/>		io.GPIOReader
Type: io.GPIOReader		Edge triggered GPIO state reader for pin <pin>
Argument	Value	Outputs:
edge	"b"	state : state
gpio_pin	23	
pull	"d"	

- In the category **media** in the ActorStore a MediaPlayer actor is available. Drag and drop it next to the GPIOReader actor.
- MediaPlayer needs to be initiated with the address of an audio file. In the example folder the audio file *<dingdong.ogg>* is available.
- Give the MediaPlayer actor the name Bell



--- Button ---

io.GPIOReader
state ▶

--- Bell ---

media.MediaPlayer
▶ play

Name:

Type: media.MediaPlayer

Argument	Value
media_file	"dingdong.ogg"

media.MediaPlayer

Play media file <mediafile>.

Inports:

play : Play <mediafile> when True

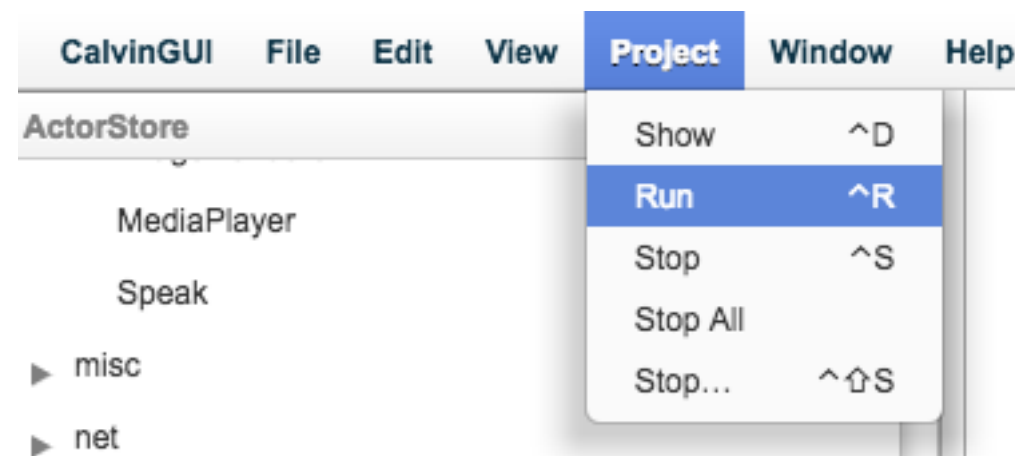
Requirements:

calvinsys.media.mediaplayer

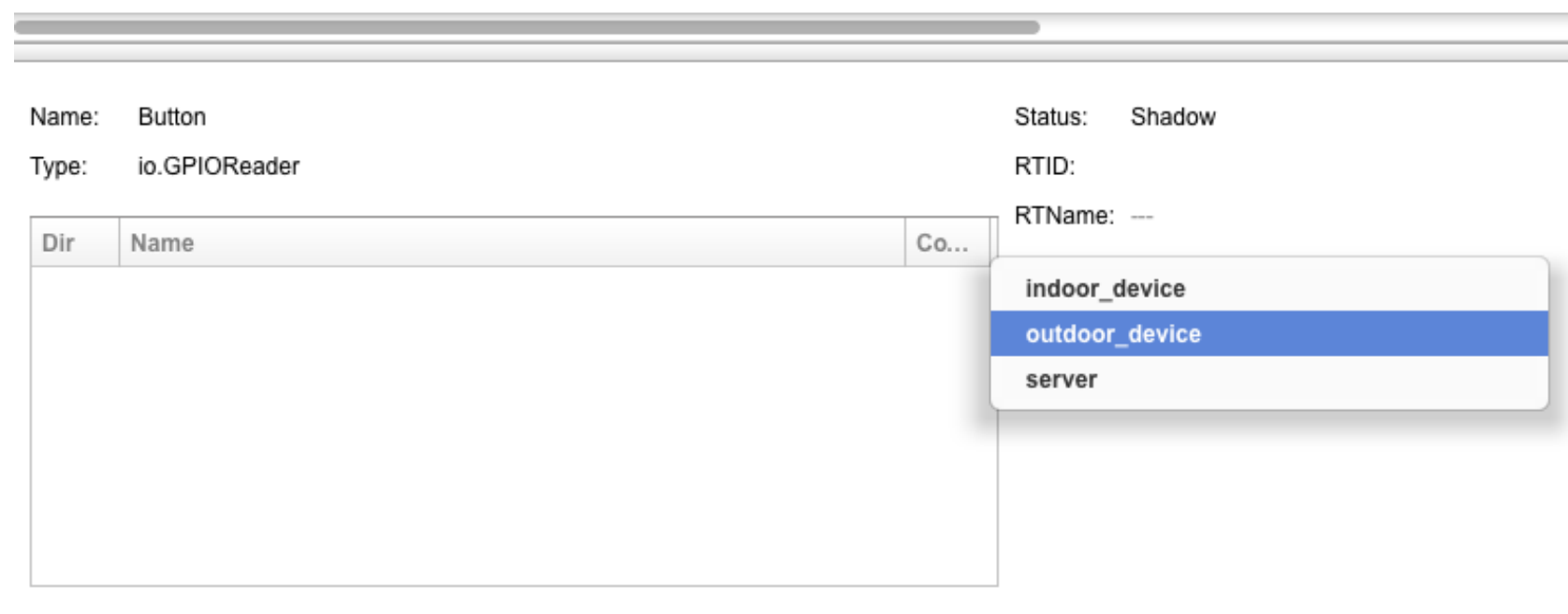
- The application is now almost ready, but for data to flow between the actors the ports need to be connected. The small grey triangles in the actor boxes represent the ports.
- Click on the out-port of the GPIOReader actor and drag a connection to the in-port of the MediaPlayer actor to connect the ports.
- The GPIOReader will forward either **0** or **1** on its outputport **state** depending on the state of the button.
- When MediaPlayer receives a token which can be evaluated to True, e.g. **1**, on its in-port **play**, the actor will start to play its media file.



- Start the application by clicking **Run** in the **Project** menu.



- The application is now running. However, the actors are not able to perform any of their tasks since the config file for the runtime *server* doesn't specify any `gpio_plugin` or `mediaplayer_plugin`. Both actors are therefor placed as shadow actors on the runtime named *server*.
- At the current state of the Calvin GUI it is not possible to distinguish whether an actor is a shadow actor or not. If you want to verify if an actor is a shadow actor, one alternative is to use **csweb**.
- For the button to work the GPIOReader actor needs to migrate to a Raspberry Pi with a physical button.
- Select the GPIOReader actor and pick **outdoor_device** in the runtime menu. The actor will migrate to the Raspberry Pi with a runtime named *outdoor_device*.

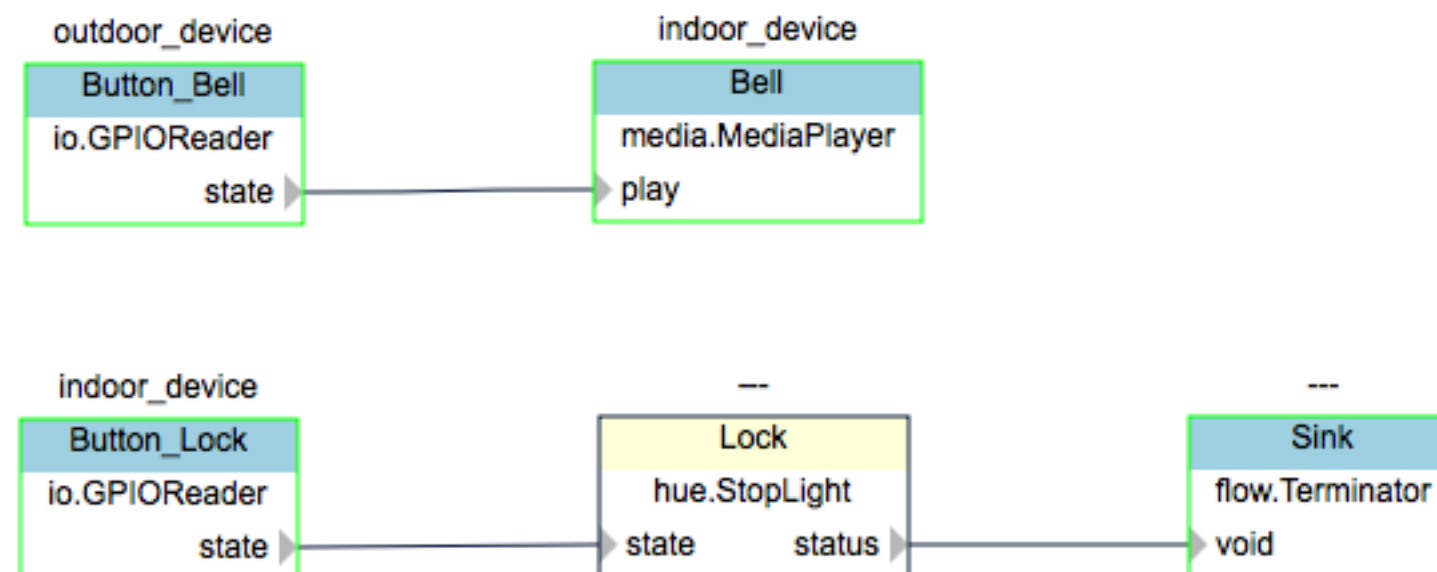


- Migrate the MediaPlayer actor in the same manner but this time to the **indoor_device**.
- If the migration was successful the labels above the actors will be updated. If labels doesn't update automatically, try clicking once in the middle of the window for the GUI to be redrawn.
- Try the application by pressing the physical button on the Raspberry Pi to which the GPIOReader actor was migrated.
- If nothing happens, please jump to the end of this manual for some tips about how to debug an application.

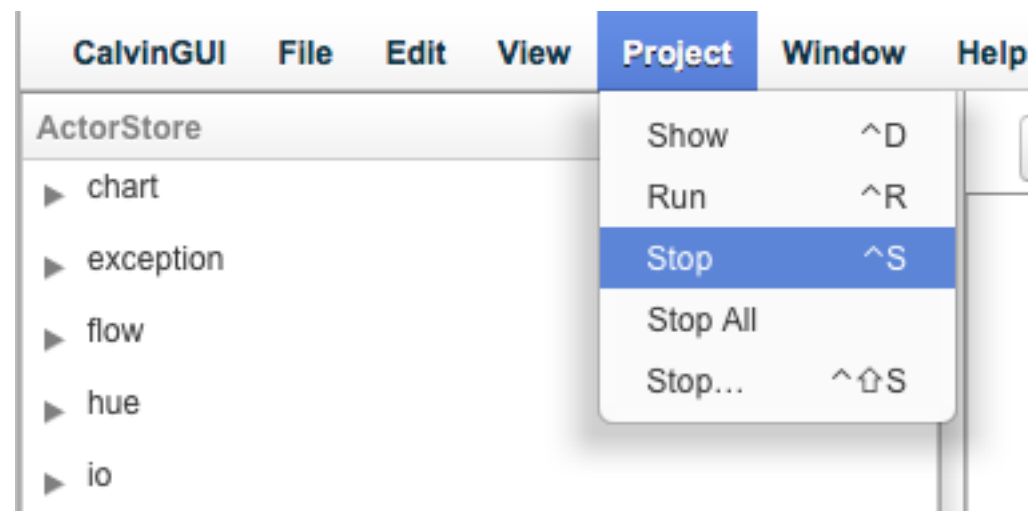


Lock

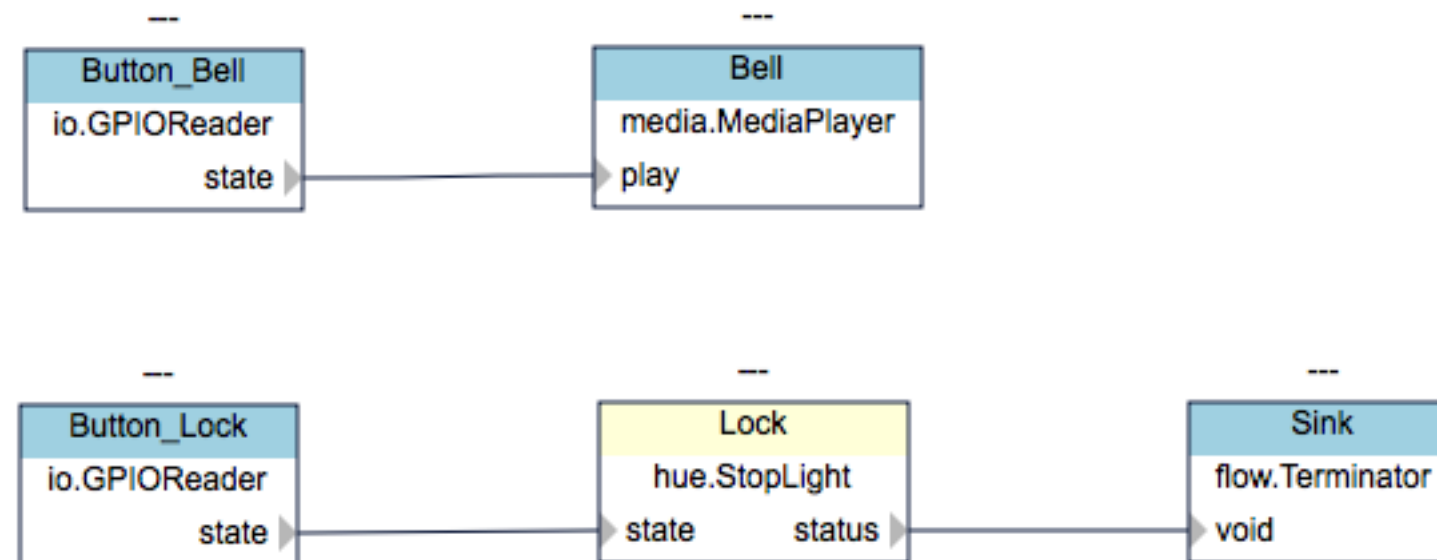
- The application will now be extended with a Philips Hue lamp acting as a lock accessible from the inside. The hue lamp will shine with a green light to indicate an open door and a red light to indicate that the door is locked.
- If no Philips Hue is available the status of the lock can be displayed in the terminal instead, see the **Lock without Hue** section instead if no Philips Hue is available.



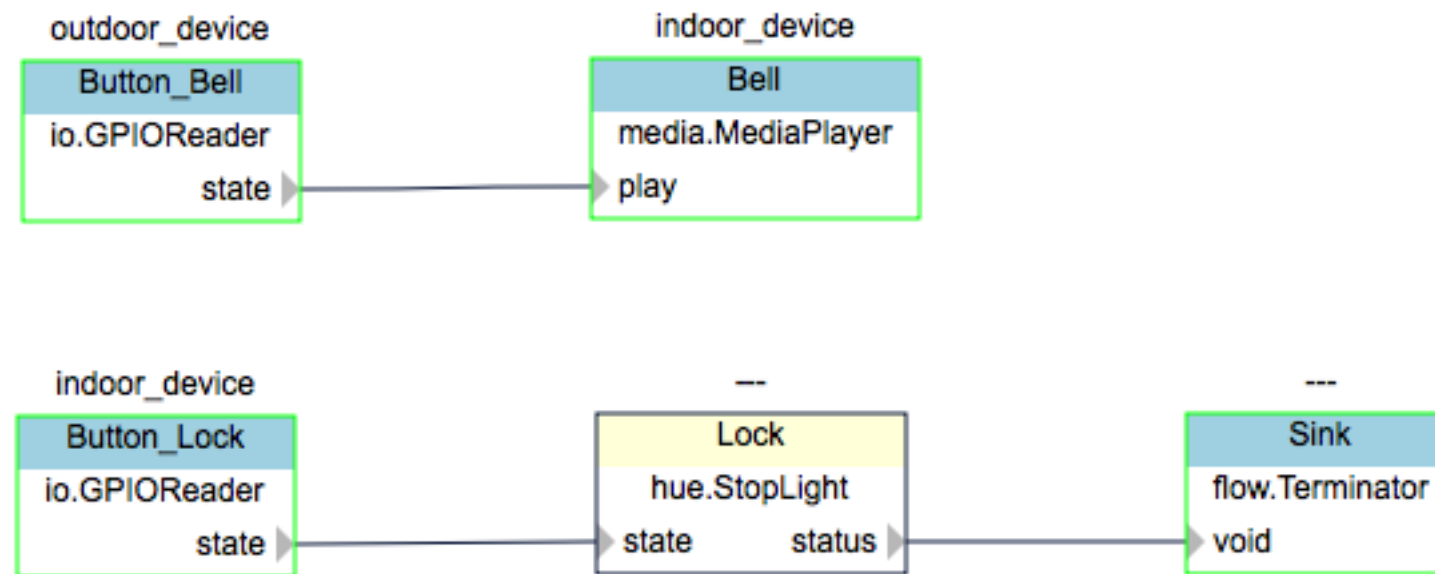
- Start with stopping the last application by choosing **Stop** in the menu **Project**



- In addition to the already present GPIOReader and MediaPlayer actors, pick the following actors and drag them out to the working area as before:
 - GPIOReader in the category **io**.
 - StopLight in the category **hue** (this is a component composed of many actors, it needs to be installed to show up in the ActorStore. Instructions are available in the README file).
 - Terminator in the category **flow**. This actor could be changed to a Log or Print actor to verify the status sent out from the Lock component.
- Initiate the actors as before and connect them as shown in the picture below.



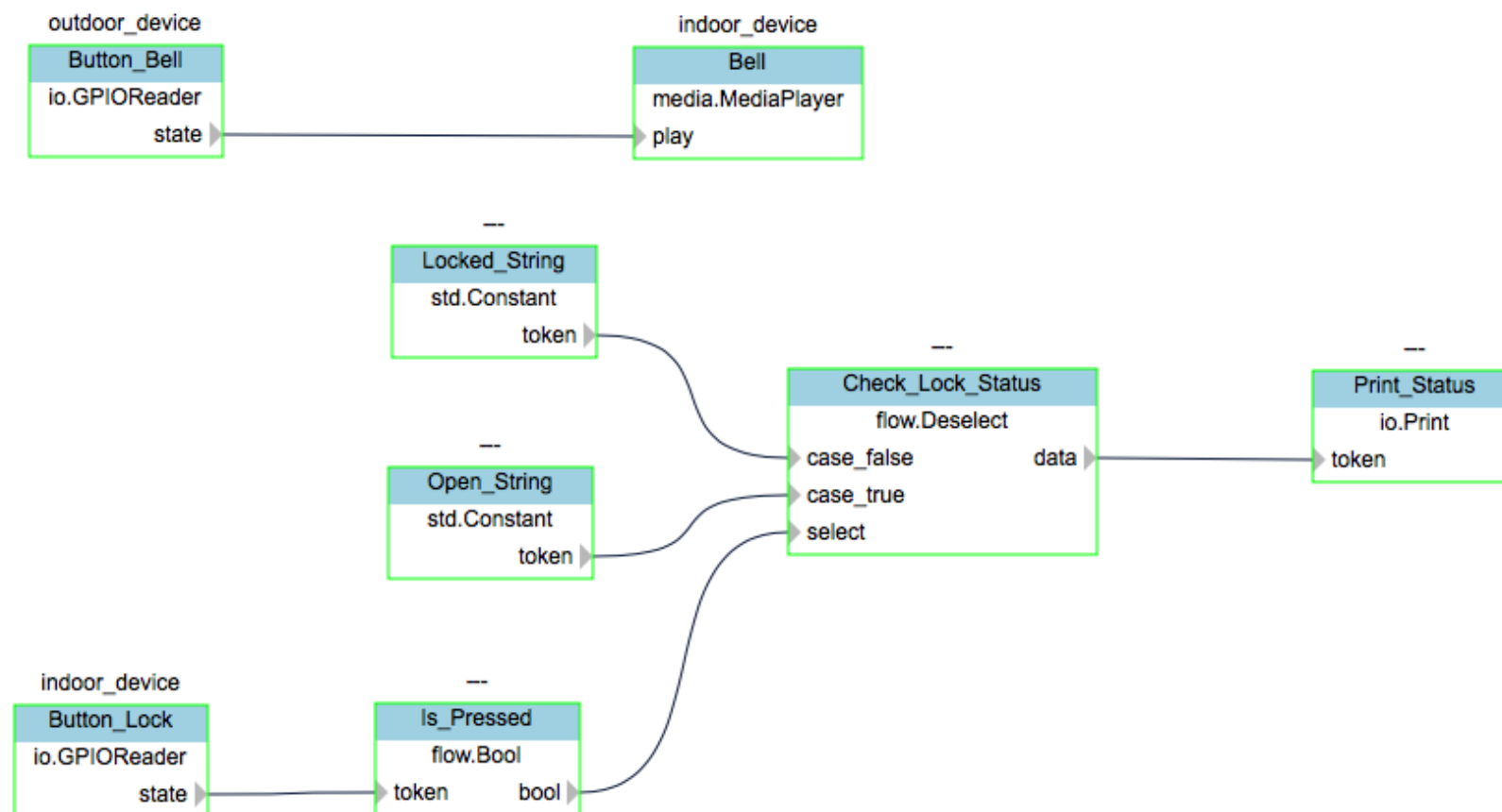
- Start the application and migrate the GPIOReader and MediaPlayer actors as stated in the labels below:



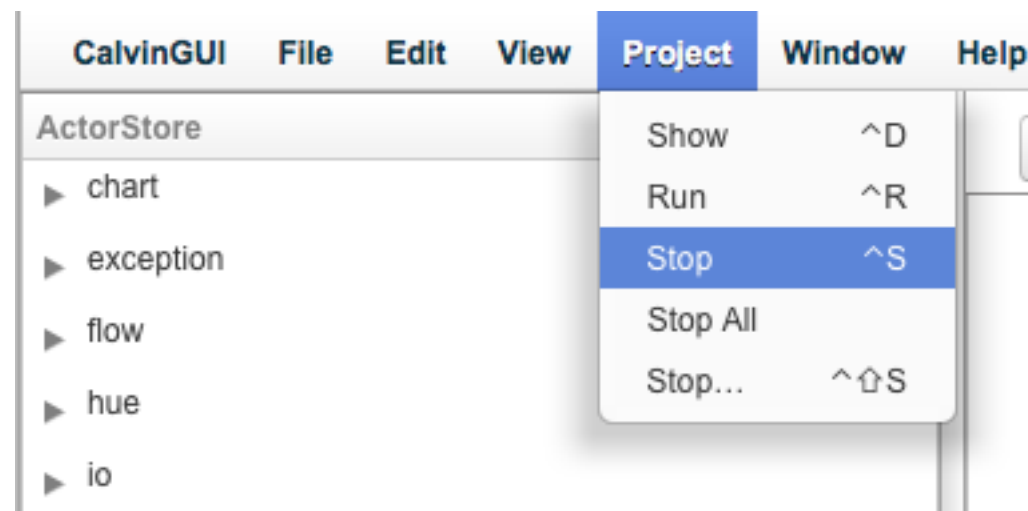
- Try it out by pressing the physical buttons on the two raspberry pis.

Lock without Hue

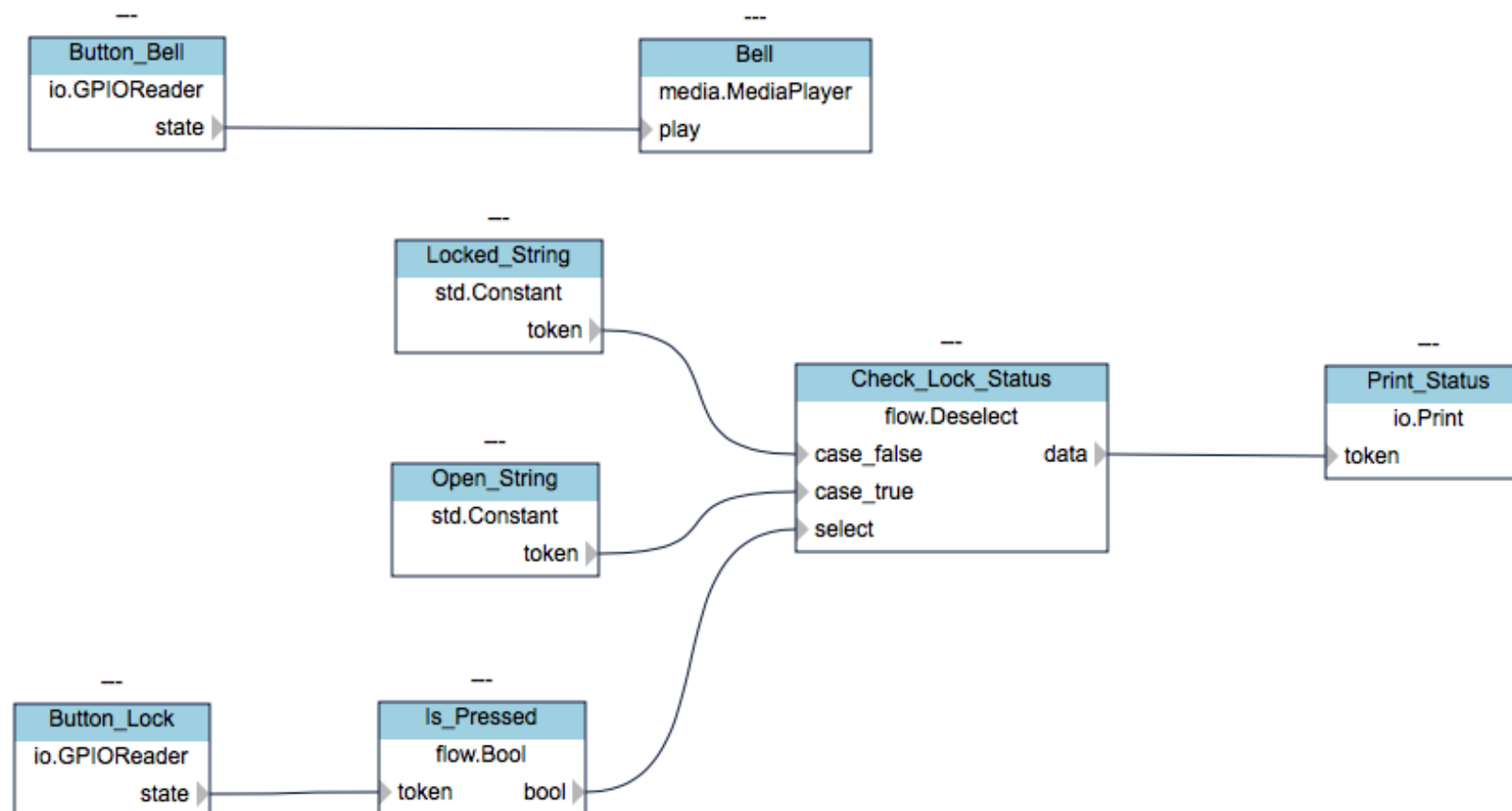
- The application will now be extended with a couple of actors simulating a lock accessible from the inside. A Print actor will write the status of the lock to the terminal.



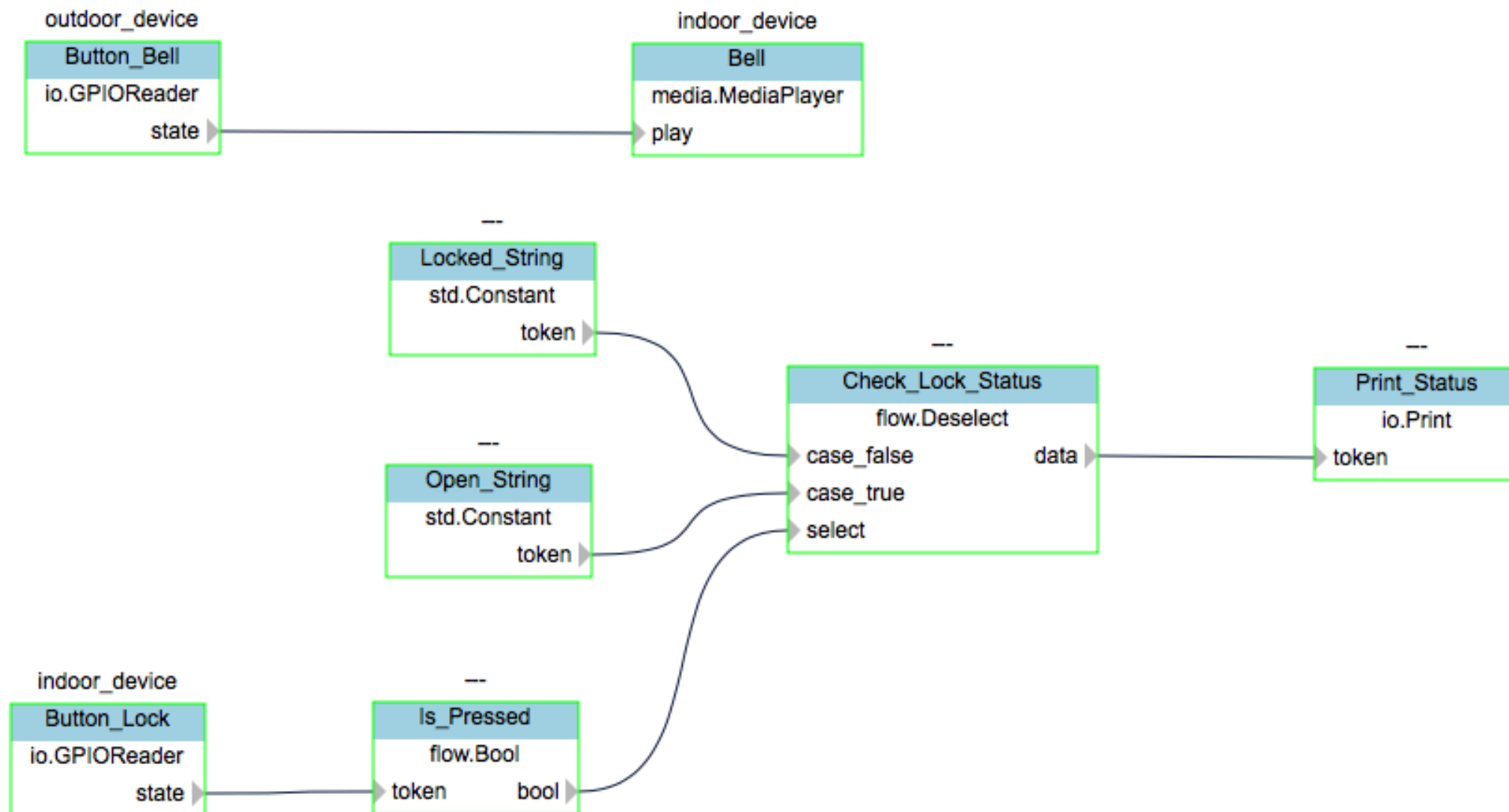
- Start with stopping the last application by choosing **Stop** in the menu **Project**



- In addition to the already present GPIOReader and MediaPlayer actors, pick the following actors and drag them out to the working area as before:
 - From the category **io**:
 - A Print and a GPIOReader actor.
 - From the category **flow**:
 - A Bool and a Deselect actor.
 - From the category **std**
 - Two Constant actors.
- Initiate the actors as before and connect them as shown in the picture below.



- Start the application and migrate the GPIOReader and MediaPlayer actors as stated in the labels below:



- Try it out by pressing the physical buttons on the two raspberry pis.

If it doesn't work

- First, make sure at least the first two scripts in the Scripts directory (part_1 and part_2) run as intended. If not, please refer to the README file and make sure all settings are correct.
- A simple way of debugging an application is to place an Identity actor between two actors in the application. The argument **dump** needs to be initiated to **true**. All tokens flowing through the Identity actor will then be logged to the terminal window before continuing to the next actor.

