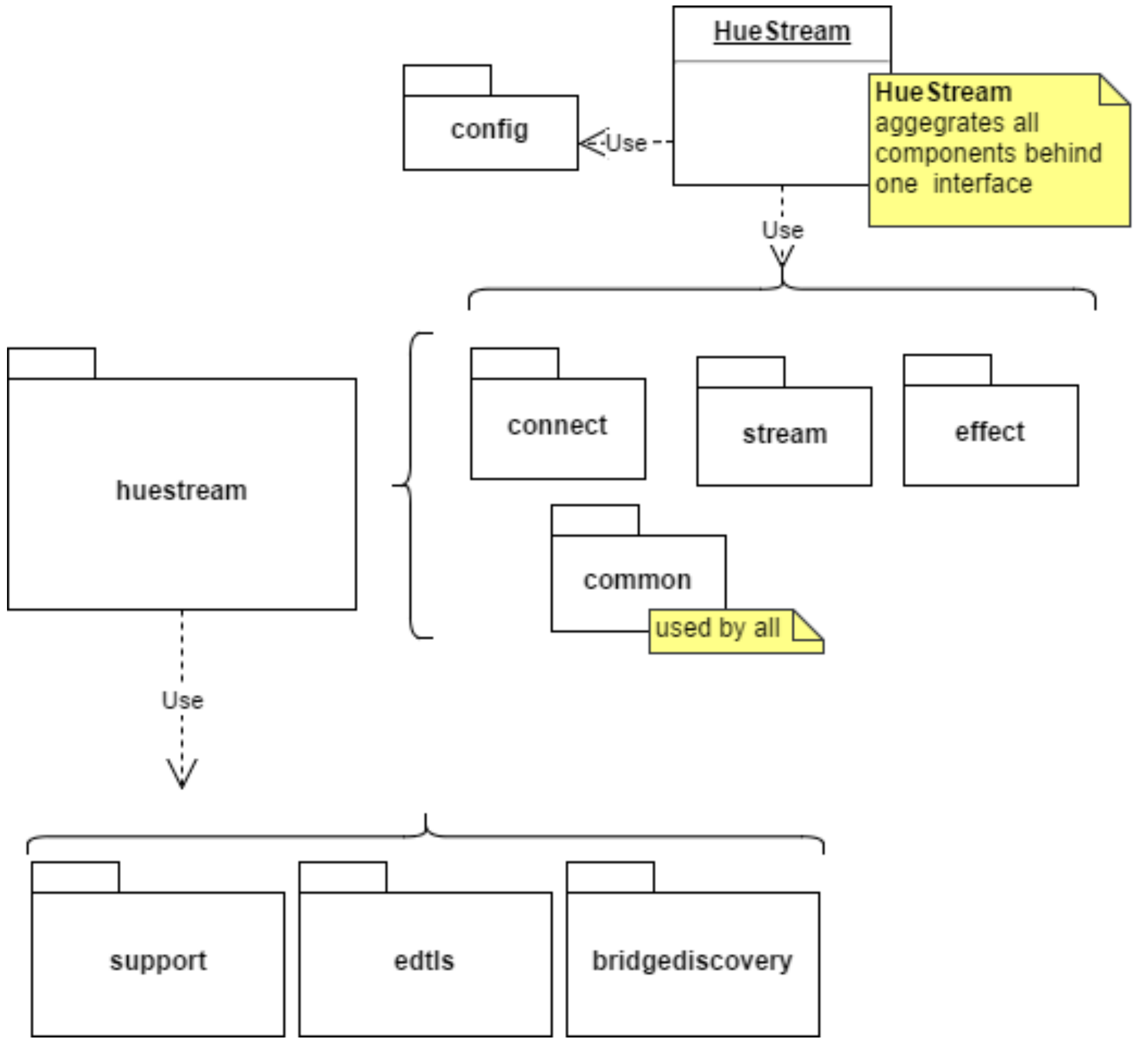
EDK concepts walkthrough

- Intro
- Overview
- Modules
    - Connect
    - Effect
        - Animation
        - Effect
        - Mixing
        - Lightscript
    - Stream
    - Config
- Swig
- Simulator
- Demo
- Discussion

These slides aim to explain the concepts of the EDK.
Details may be slightly outdated with the actual source code, all code shown is pseudo-code.

Overview



Responsibilities

- Bridge connection flow
- Secure streaming implementation
- Light effect engine

Portability

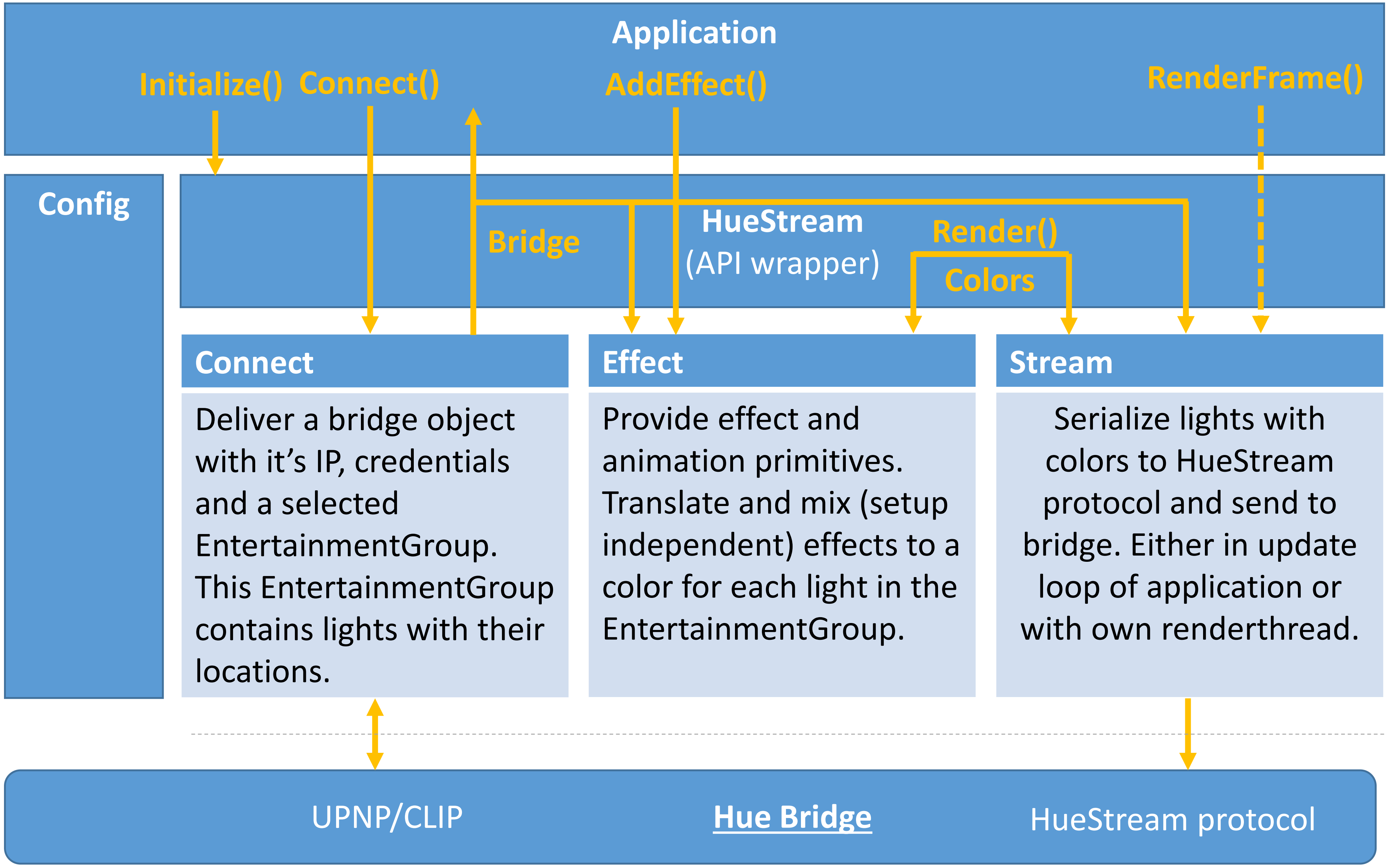
- C++ 11
- MSVC [2015+], Clang (LLVM) [3.3+], GCC/MinGW [4.9+]
- Windows (PC / Xbox One), PlayStation 4, Nintendo Switch, Android, iOS, Linux/MacOS
- Wrappers (automatically generated): C#, Java, Objective-C, (Python)
- No RTTI, No Exceptions
- Published source code

Responsibilities of Connect, Stream and Effect modules

HueStream API wrapper connects modules together to a single API. Config is used to inject settings. Modules could be replaced e.g. a game using its own animation engine.

Bridge Object

- Name
- ID
- IP
- Credentials
- EntertaimentGroups
  - Lights
    - Location
    - Color
- SelectedGroup
- ...



Connect

Deliver a bridge object with it's IP, credentials and a selected EntertainmentGroup. This EntertainmentGroup contains lights with their locations.

Effect

Provide effect and animation primitives. Translate and mix (setup independent) effects to a color for each light in the EntertainmentGroup.

Stream

Serialize lights with colors to HueStream protocol and send to bridge. Either in update loop of application or with own renderthread.

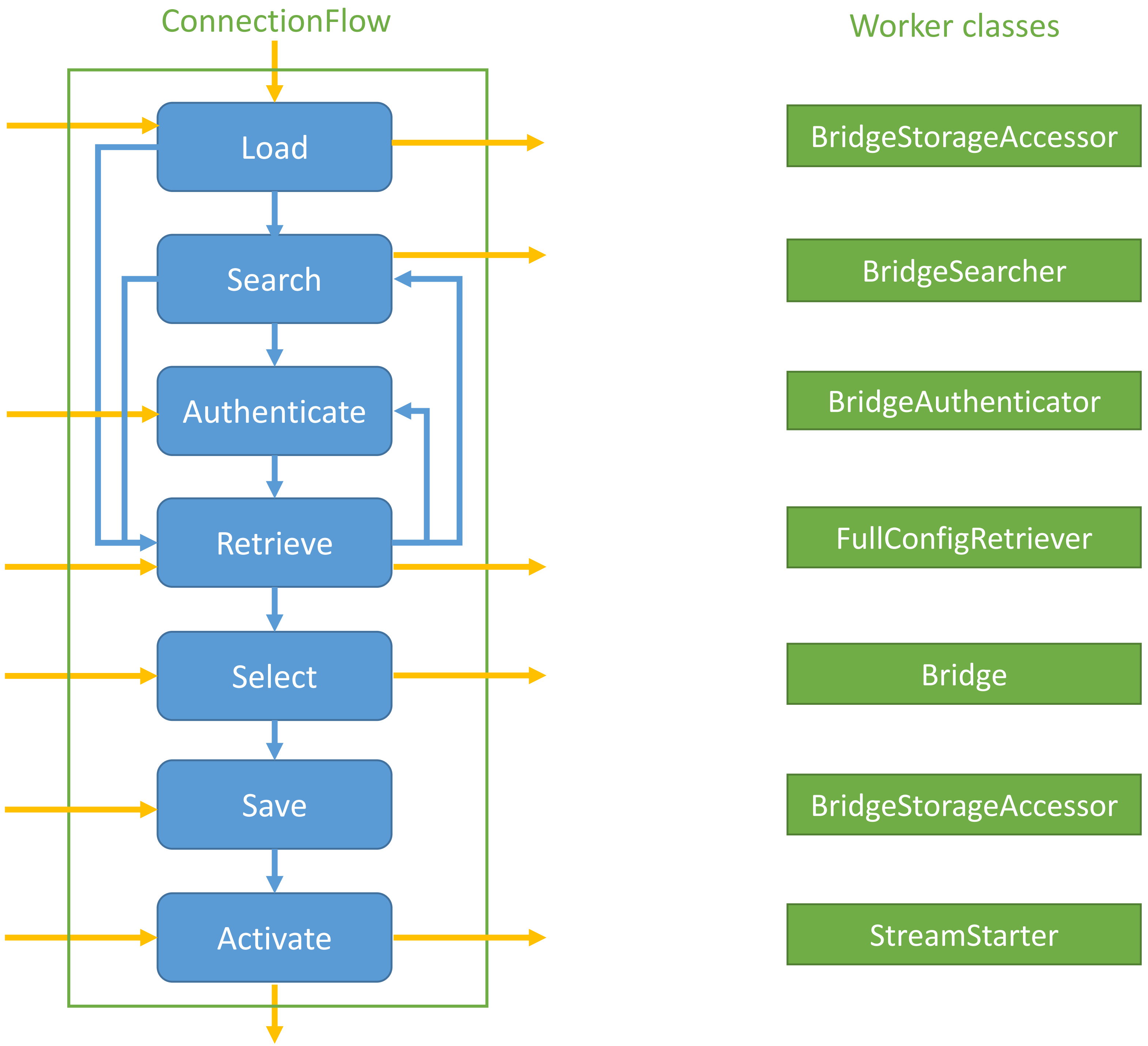
Connect

API's (Sync vs Async)

- Connect
- Connect background
- Connect manual ip
- Connect full manual
- Select group
- Reset
- Separate load
- Separate activate
- Abort

Cases handled

- Retry search with ipscan
- Authorization lost
- IP address changed
- Difference V1 and V2 bridge
- Already streaming
- No group selected
- Invalid group selected
- Invalid model
- Invalid SW version
- Bridge not found
- No new bridge found
- ....



Connect: Feedback

Via asynchronous callback

Callback provides a FeedbackMessage which indicates:

- The type of ongoing **request**: GetRequestType()
- An enum **id**: GetId()
- A **tag** (or string id) used for translation: GetTag()
- A message **type**: GetType()
- If the message type is USER, a **user message** string in the language the EDK is configured in: GetUserMessage()
- A **debug message** string: GetDebugMessage()

Note that instead of a callback, the application can choose to implement the IFeedbackMessageHandler interface and set it via RegisterFeedbackHandler(FeedbackMessageHandlerPtr handler).

Via synchronous request

- GetConnectionResult()
- GetLoadedBridge()->GetStatus()

Effect

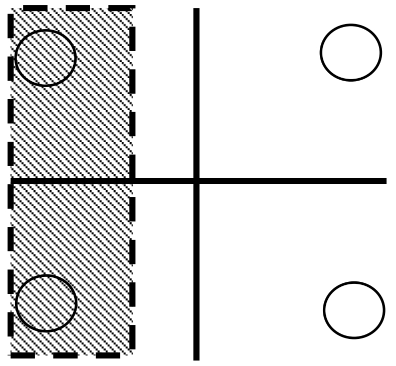
Effect
Assign colors to lights using a mapping which is **independent of a specific light setup**

Animation
A mapping of time to a ‘value’, which can be used by effects to animate their properties

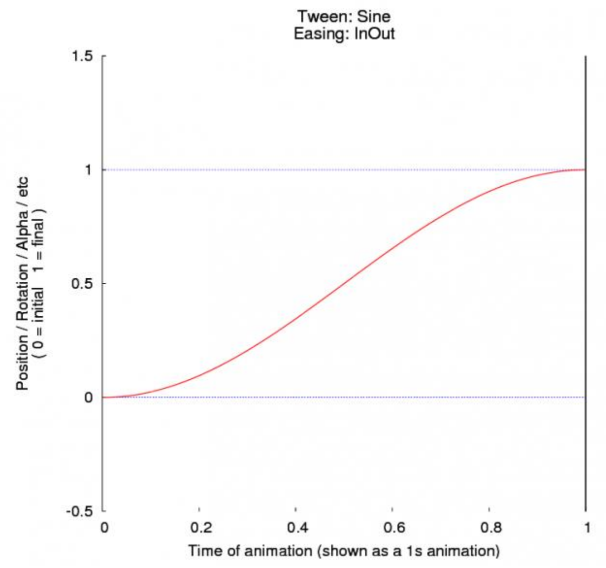
Mixer
Mix different effects by layer/transparancy to per frame render the final color for each light

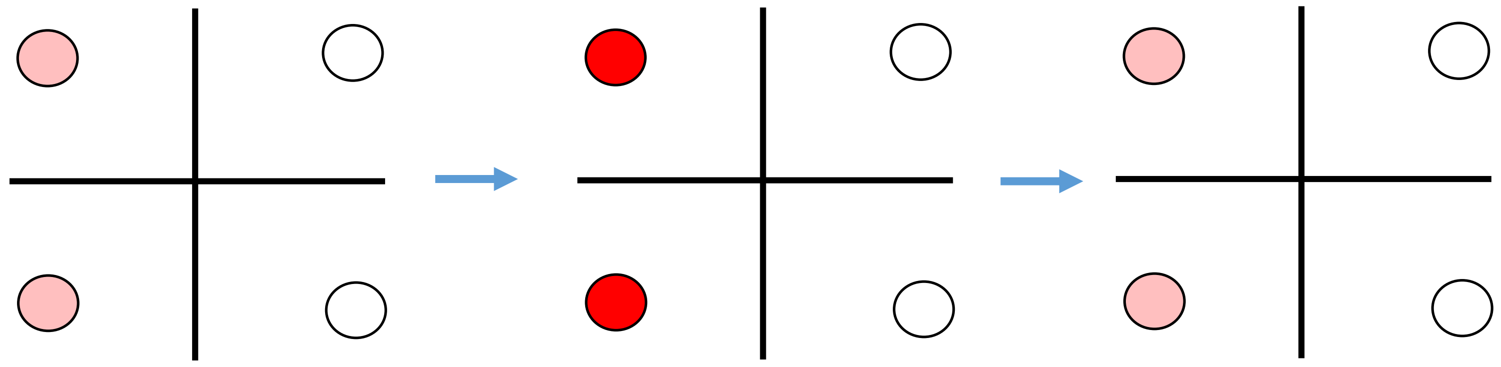
Lightscript
Provide a way to bind multiple effects to a timeline and import/export such lightscripts

First effect



```
Effect = hueStream.CreateEffect<AreaEffect>("leftRedSine",0)
effect.addArea(LEFT)
Sequence sineAnimation(INF)
sineAnimation.append(Tween(0,1,1000,SINE_INOUT))
sineAnimation.append(Tween(1,0,1000,SINE_INOUT))
effect.setColorAnimation(sineAnimation,Constant(0),Constant(0))
hueStream.addEffect(effect)
effect.enable()
```

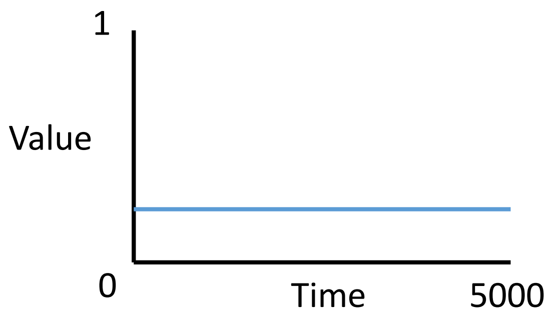




Animation type examples

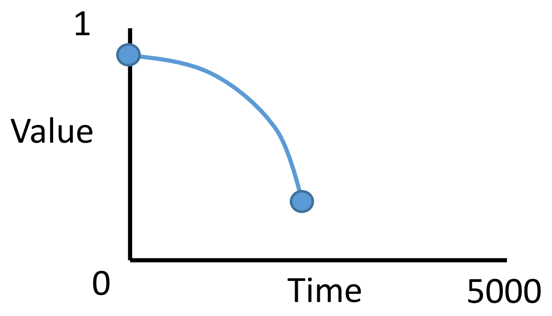
Constant

```
Constant constant(0.25)
```



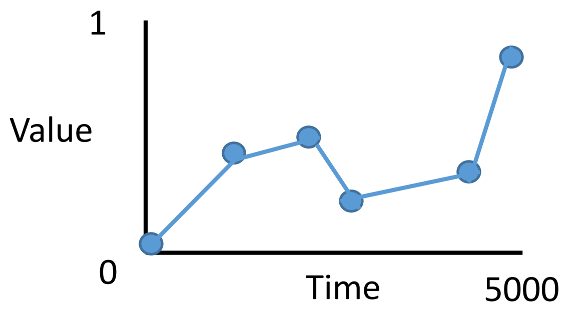
Tween

```
Tween tween(0.9, 0.25, 2200, QUAD)
```



Curve

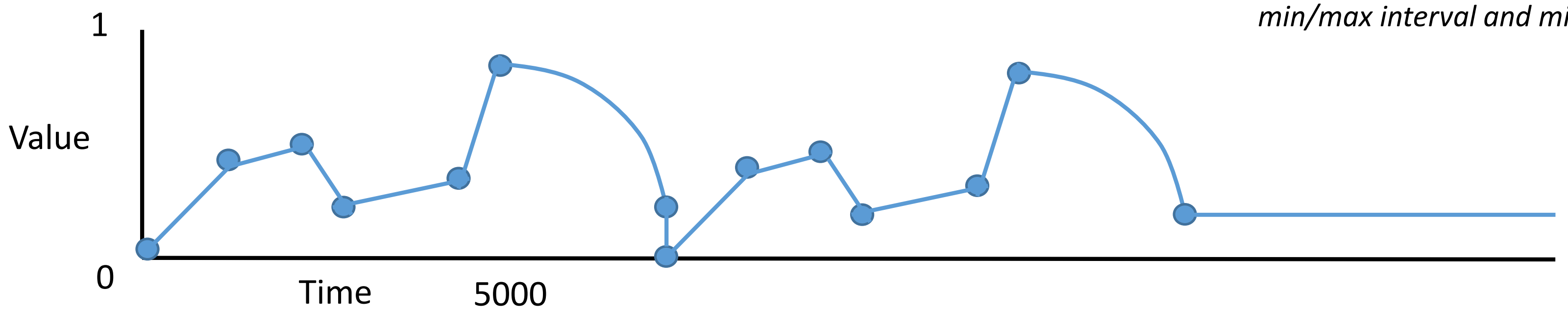
```
Curve curve
curve.add(Point(0,0))
curve.add(Point(1500,0.4))
curve.add(Point(2000,0.5))
curve.add(Point(2500,0.25))
curve.add(Point(4500,0.3))
curve.add(Point(5000,0.9))
```



Sequence

```
Sequence seqCurveTween
seqCurveTween.setRepeat(1)
seqCurveTween.append(curve)
seqCurveTween.append(tween)

Sequence seqTotal
seqTotal.setRepeat(0)
seqTotal.append(seqCurveTween)
seqTotal.append(constant)
```



Randomizer

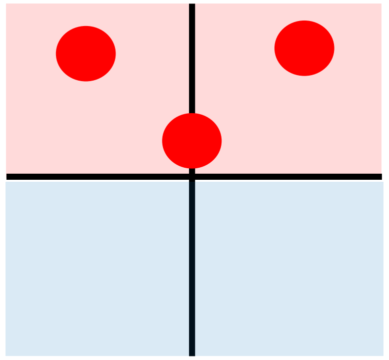
There's one more animation type: 'RandomTween', which generates random tweens between specified min/max interval and min/max values.

Effect type examples

AreaEffect

```
AreaEffect frontRed
frontRed.addArea(FRONTHALF)
frontRed.setColorAnimation(Constant(1),Constant(0),Constant(0))

AreaEffect backBlue
backBlue.addArea(BACKHALF)
backBlue.setColorAnimation(Constant(0),Constant(0),Constant(1))
```



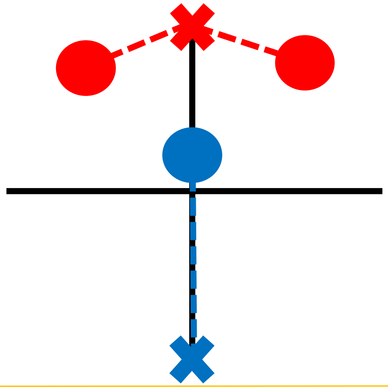
AreaEffect will play on all lights in a given area. This also means if there's no light in the area, the effect won't be visible.

MultiChannelEffect

```
Channel frontRed
frontRed.setLocation(Location(0,1))
frontRed.setColorAnimation(Constant(1),Constant(0),Constant(0))

Channel backBlue
backBlue.setLocation(Location(0,-1))
backBlue.setColorAnimation(Constant(0),Constant(0),Constant(1))

MultiChannelEffect effect
effect.addChannel(frontRed)
effect.addChannel(backBlue)
```

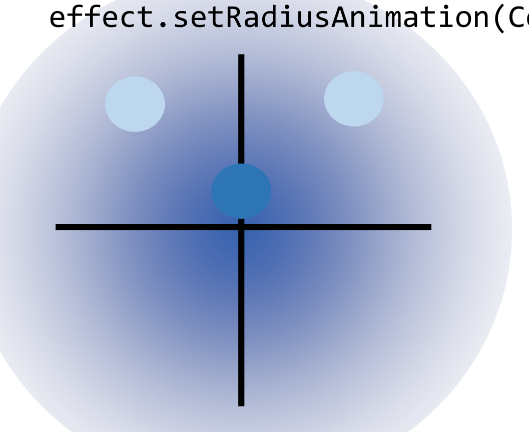


MultiChannelEffect will try to distribute the light channels (compare to e.g. audio channels) evenly over the available lights, prioritizing lights closest to the channel location.

These 4 effect base types serve as a good start point but with these examples an application could design their own types.

LightSourceEffect

```
LightSourceEffect effect
effect.setColorAnimation(Constant(0),Constant(0),Constant(1))
effect.setLocationAnimation(Constant(0),Constant(0))
effect.setRadiusAnimation(Constant(1.5))
```

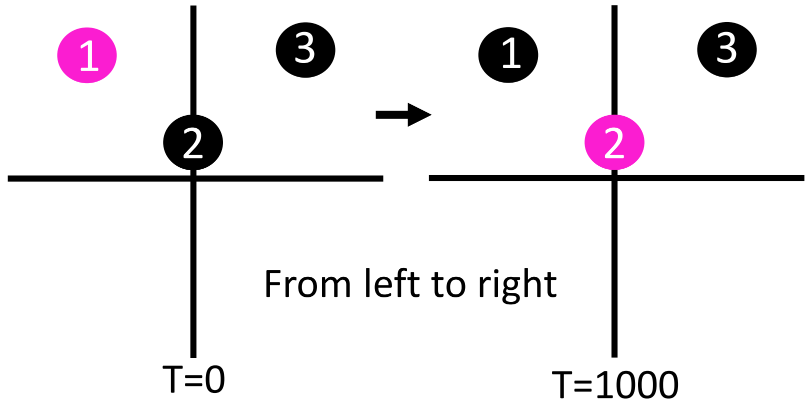


LightSourceEffect will map a virtual light source to actual lights such that lights close to the light source are more strongly influenced than lights furhter away from the light source

LightIteratorEffect

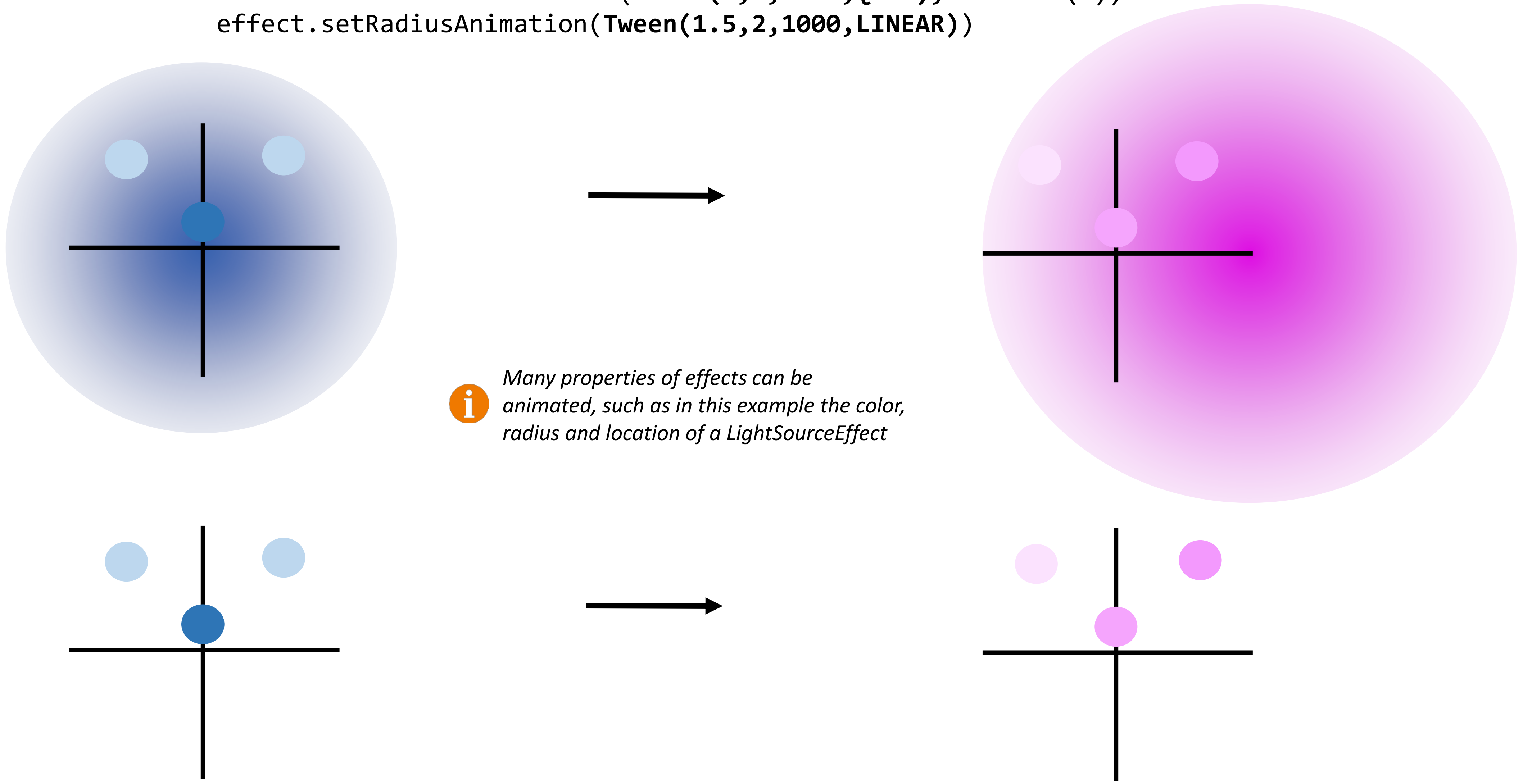
```
LightIteratorEffect effect
effect.setColorAnimation(Tween(1,1,1000,LINEAR),Constant(0),Constant(0))
effect.setOrder(LEFTRIGHT)
effect.setMode(CYCLE)
effect.setOffset(1000)
```

LightIteratorEffect will iterate an animation over individual lights with a certain offset, order and mode. This means that the total duration of an iteration over all lights depends on the number of lights in the setup.



Combining effect with animation example

```
LightSourceEffect effect
effect.setColorAnimation(Tween(0,1,1000,LINEAR),Constant(0),Constant(1))
effect.setLocationAnimation(Tween(0,1,1000,QUAD),Constant(0))
effect.setRadiusAnimation(Tween(1.5,2,1000,LINEAR))
```

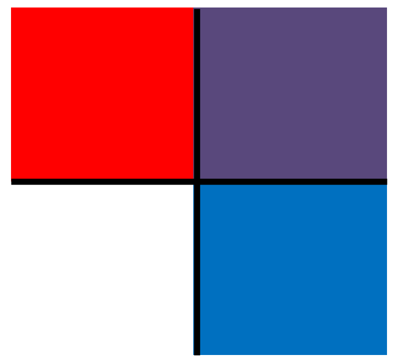


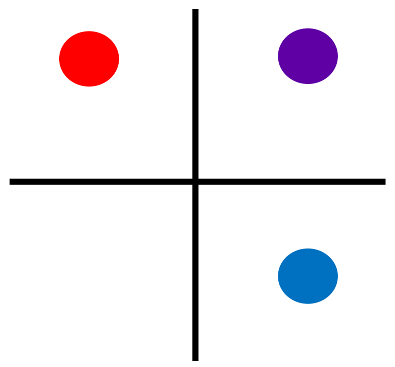
Many properties of effects can be animated, such as in this example the color, radius and location of a LightSourceEffect

Effect mixing example

```
AreaEffect frontRed(“background”, 0)
frontRed.addArea(FRONTHALF)
frontRed.setColorAnimation(Constant(1),Constant(0),Constant(0))
frontRed.setOpacityAnimation(Constant(1))

AreaEffect rightBlue(“foreground, 1)
rightBlue.addArea(RIGHTHALF)
rightBlue.setColorAnimation(Constant(0),Constant(0),Constant(1))
rightBlue.setOpacityAnimation(Constant(0.65))
```





Lightscript with timeline example

- Lightsript
  - Metadata
    - Name
    - Ideal setup
  - Script
    - For earch layer a list of actions
      - Action is just a container for a playable effect
        - Adding a start position and optional explicit end position (vs implicit by effect)
        - Injecting a player which has a timeline as timeprovider (instead of ‘real’ time)

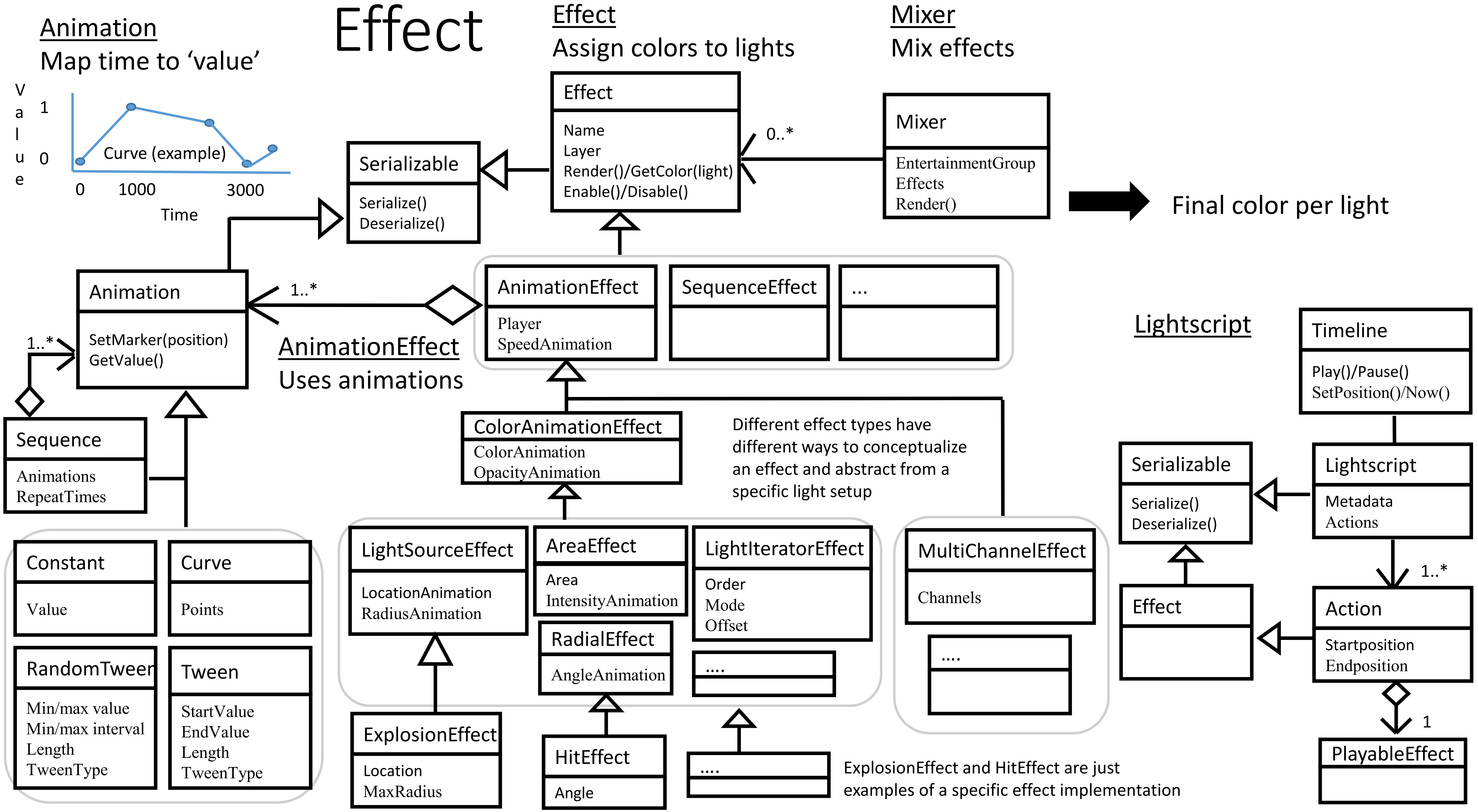
- Serialize and deserialize (JSON)

- Can be bound to timeline
  - Timeline can play/pause by itself
  - Position can be fully controlled by application → frame by frame visible on lights
  - Or a combination where it plays by itself but regularily synced

- Example lightscript with 3 actions

script.json

Effect



Different effect types have different ways to conceptualize an effect and abstract from a specific light setup

ExplosionEffect and HitEffect are just examples of a specific effect implementation

STREAM component

External API to construct the frames for streaming

- Render single frame
- Render continuously in render thread

Serialize into frames of hue-stream-protocol message

- Rendering of a frame consists of serializing the selected entertainment group into

Send message through a connector class

- Mbedtls connector sends hue-stream-protocol messages through mbedtls stack to the HUE bridge
- UDP connector send (plain) hue-stream-protocol messages through UDP socket (used by simulator)

Configuration

- App name, platform
- Language, region
- Use animation engine
- Use renderthread
- Auto start at connection
- Frame rate
- Color Mode
- UDP vs DTLS connector
- Inject own implementations
    - BridgeStorageAccessor
    - HTTPRequest
    - EntropyProvider
    - Translator
    - etc

Simulator

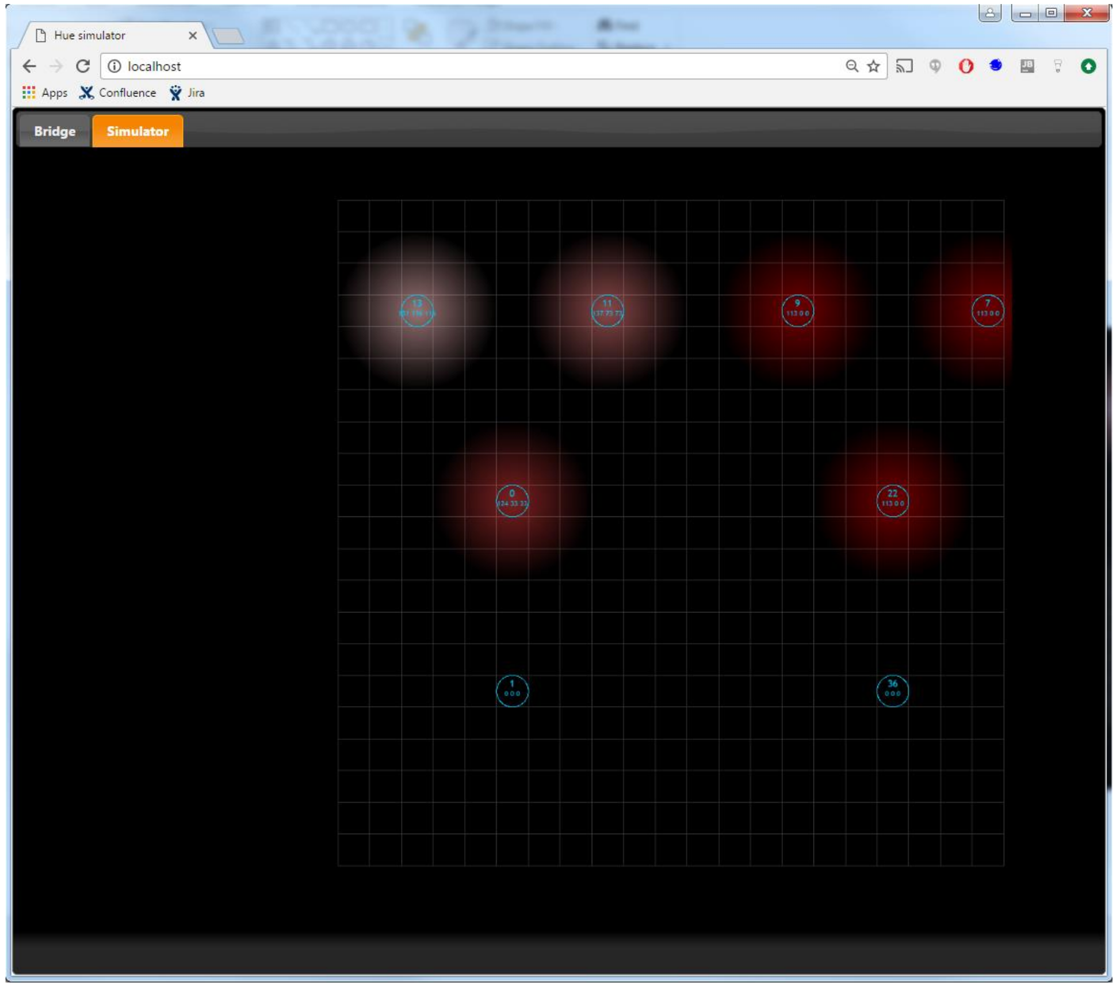
NodeJS based simulator

Development tool to experiment with effects and different setups without hardware.

- Server
    - Host client page (see client)
    - Very minimal CLIP implementation
        - Full config
        - Entertainment group configuration
        - Pushlinking
    - UDP server to receive huestream protocol messages and deserialize messages into javascript stream objects.
    - Host websocket server for client page pushing javascript objects
- Client
    - Connects to server websocket to receive stream objects
    - Renders stream objects

Running the simulator

- (once) Install Node.js
- (once) Run install.cmd/sh
- Run start.cmd/sh
- Localhost in browser



Examples

C++

- huestream_example_console: small console based example running on Windows/Linux
- huestream_example_gui_win: more extended Windows-only GUI based example

Wrappers (only if BUILD_WRAPPERS=ON)

- huestream_csharp_managed: generates Visual Studio C# project example in output directory
- huestream_java_native: generates Eclipse Java project example in output directory