Hue EDK Effect Creation Application Note

1. Context

There are three main components in EDK:

1. Hue bridge discovery and linking
2. Streaming API wrapper, including authentication and encryption (DTLS) implementation
3. Light Effect-rendering Engine

This application note focusses on the light effect-rendering engine which provides a framework to create, play and mix various types of light effects.

The effect engine:

- Provides a framework for the application to create and add light effects
- Uses the user's light setup retrieved by the bridge connection component to map generic light effects to a specific light setup
- Renders a light frame with the most recent color for each light on periodic requests from the streaming component

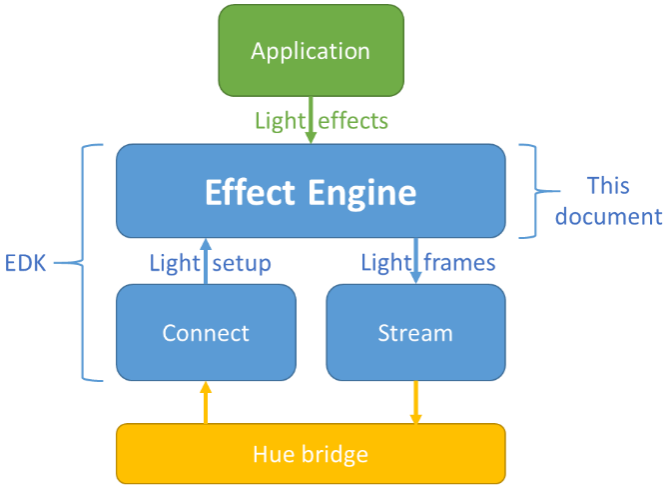


FIGURE 1: HUE LIGHT EFFECT ENGINE CONTEXT

After reading this application note, you will understand how to create a wide range of light effects using the Hue light effect engine.

2. Conventions

The Hue light effect rendering engine – and by extension this document – uses the following conventions:

- Colors are defined in Red, Green, Blue, Alpha/Opacity (RGBA) color components ranging between 0 and 1
- Times are defined in milliseconds
- Speed is defined as a multiplication factor over the default speed of effect animations (i.e. default speed is 1)
- Locations are defined relative to a user's approximated room which spans from -1 to 1 in both x (left to right) and y (back to front), see Figure 2.

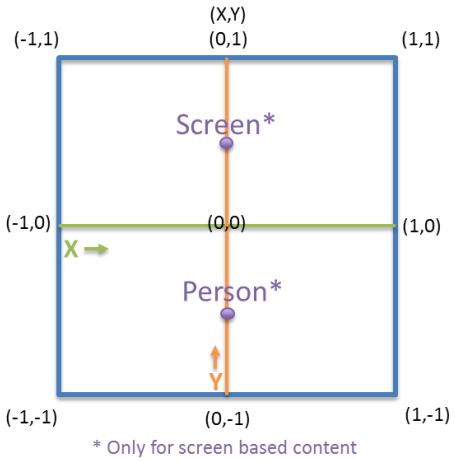


FIGURE 2: DEFINITION OF ENTERTAINMENT LIGHT SETUP SPACE

3. Hue Light Effect-rendering engine concepts

Let's start with an overview of the four main concepts used within the Hue light effect engine:

3.1. Effect

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

We define a light effect as a coherent way of assigning colors to lights. Two typical examples of a light effect are an explosion effect and a running light. The EDK will provide:

- A couple of fully worked out light effect examples
- Multiple effect type primitives which provide a basis for creating customized light effects
- The possibility to create your own effects fully from scratch

In the end, the result of any light effect is to assign an RGBA color to every light at any moment in time.

The main challenge for designing useful light effects is to make sure they work indepent of a specific light setup. The designer may have an ideal setup in mind where the user for example has a lamp in the front left, a lamp in the front right and a lamp in the center back. But because of the nature of home lighting, different users will always have diffrent light setups. As such it is important that a light effect aims to map as well as possible to any light setup, instead of depending on a single specific setup. Examples of how this can be done will be provided in section 5.

3.2. Animation

A mapping of time to a 'value', which can be used by effects to animate their properties.

An often recurring requirement for creating a light effect is for a certain property to change over time. This can be a color or brightness, but for example also a radius or location. The goal of an animation is to support this by providing different ways to define how a certain value changes over time. Animations will be further explained with examples in section 6.

3.3. Mixer

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

In the effect engine, multiple effects can be enabled and affect lights at the same time. Therefore, these effects must be mixed. The light effect rendering engine contains a single effect mixer. Every effect which follows the effect interface can be added to the mixer. Also, every effect has an assigned layer, and every

color includes an opaqueness property. Based on this, all effects which are added to the mixer and enabled, will be automatically mixed according to their layer and opacity. The outcome is a single color for each light per light frame. An example will be provided in section 8.

3.4. Lightscript

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

Effects can be triggered at any time based on any input the developer deems relevant. However, one typical use cases for triggering light effects is to schedule them on specific times. To support this, the engine provides the concept of a lightscript. A lightscript contains actions, which combine any type of effect with a start time and optional end time. A lightscript can be bound to a timeline for playback and can be imported and exported to JSON. More information on lightscripts can be found in section 9.

4. Creating a first light effect

There are multiple ways to create a new light effect:

- Fully from scratch by deriving from the effect base class
- Using an effect type primitive provided by the light effect engine
- Instantiating a predefined effect from the effect library

In the example in Figure 3, we will create an effect based on a simple effect type primitive called 'AreaEffect'. Pseudocode is used for the sake of readability and applicability to different programming languages.

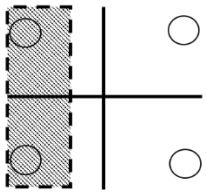
First, we create an effect object with the name "leftRedSine" on layer 0 (lowest layer). We define that this effect should affect the predefined area called 'LEFT', which means all lights in the left of the room will be affected. The AreaEffect primitive will be further explained in section 5.1.

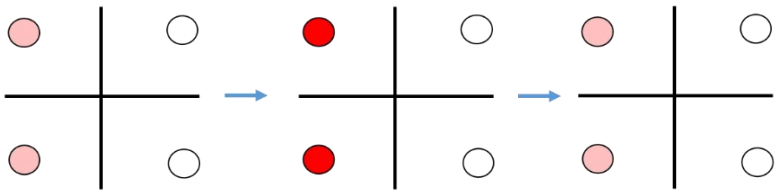
Then we create an animation object which is an infinitely repeating sequence of two so-called 'tweens': one tween describes a sinus trajectory from 0 to 1 in 1000ms, and another tween does exactly the opposite. The Tween animation type will be further explained in section 6.3.

Next, we assign this animation to the red color channel of the previously created effect. The green and blue color channels are assigned a constant zero value.

Last, we add the effect to the mixer and enable it. The result is that all lights in the left of the room will play a red breathing effect.

```
AreaEffect effect("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::GetInstance().addEffect(effect)
effect.enable()
```





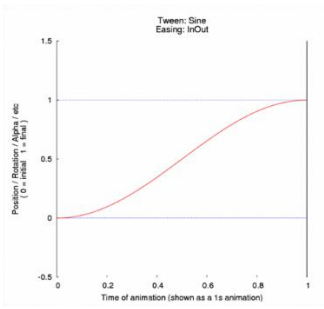


FIGURE 3: EXAMPLE OF CREATING A CUSTOM LIGHT EFFECT

5. Effect type primitives

The effect rendering engine provides four main effect type primitives. These aim to make it easier to create various types of custom effects which are independent of a specific light setup. Apart from these four, more effect type primitives can be added by developers themselves.

5.1. AreaEffect

AreaEffect will play an animation on all light sources in a given area (or multiple areas). Areas are defined as rectangles by a top left and a bottom right corner point. An important property of an AreaEffect is that if a specific setup has no light in a certain area, the effect won't be visible. This may be desired in case it is better to not show an effect at all, than to show it on the wrong location.

An example of an AreaEffect is shown in Figure 4 where two AreaEffects are specified. One is a constant red color on the front half and one is a constant blue color on the back half. In this case, all three lights are located in the front half, so all lights will color red and none will color blue. If there would have been lights located in the back half then they would have colored blue.

```
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))
```

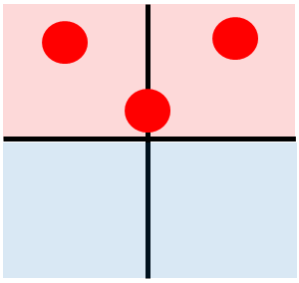


FIGURE 4: EXAMPLE AREAEFFECT MAPPING IN A POSSIBLE LIGHT SETUP

An example usage of an AreaEffect could be indicating in a game from which area a character is being hit. When e.g. a character is hit from the left it is important to only play an indication effect on the left and never on the right as this would be confusing.

5.2. MultiChannelEffect

A 'Channel' in the context of the Hue light effect engine is color animation which is tied to a specific location. Compare this to e.g. 5.1 surround sound channels in audio. MultiChannelEffect will try to distribute multiple light channels evenly over the available light sources, prioritizing light sources closest to the channel location. Notice the difference with an AreaEffect: even though a channel has a certain location, the MultiChannelEffect tries to make as many channels visible as possible, even if a channel does not have lights in close proximity to the channel location.

For example, suppose an effect creator created two light channels for an audio script, one channel in the 'Front' and another in the 'Back'. In that case he creates two Channels: one for the front (Location(0,1)) and one for the back (Location(0,-1)) and adds them to the multichannel effect. As indicated in the example in Figure 5, there is a channel with a constant red color in the front and a channel with a constant blue color in the back. Even though all lights in this setup are located closer to the red channel, still one light is mapped to the blue channel.

```
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)
```

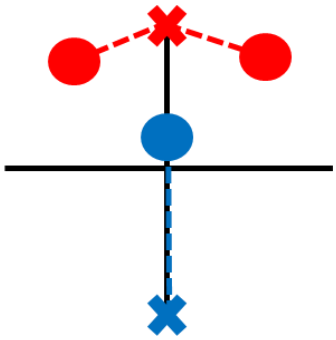


FIGURE 5: EXAMPLE MULTICHANNELEFFECT MAPPING IN A POSSIBLE LIGHT SETUP

An example usage of a MultiChannelEffect is a light show for a song which consists of four channels (e.g. frontleft, frontright, backleft, backright). Here it is important for the experience that as many channels as possible are visible, even though all lights may be located in e.g. the front right quadrant.

5.3. LightSourceEffect

LightSourceEffect will map a virtual light source to actual lights such that lights close to the virtual light source are more strongly influenced than lights further away from the virual light source. LightSourceEffects have next to an animation for color also an animation for position and radius. Note the difference with the previous effect types: in both AreaEffect and MultiChannelEffect, a light either belongs to an area or channel or not, whereas for the LightSourceEffect it's a more gradual relation: the closer to the virtual source the more strongly a light is influenced by the source.

In the example depicted in Figure 6 there is a LightSourceEffect with a constant blue color, constant location in the middle of the room and constant radius of 1.5. Lights in the actual setup which are located close to the virtual light source in the middle have a strong blue color whereas lights further away from the middle have a less strong blue color.

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

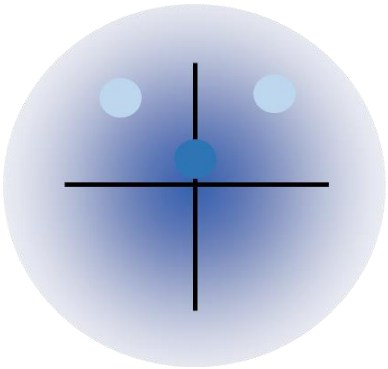


FIGURE 6: EXAMPLE LIGHTSOURCEEFFECT MAPPING IN A POSSIBLE LIGHT SETUP

The LightSourceEffect is used if the creator wants all lights within a certain radius to contribute depending on their distance to the location of the effect. Example usages of LightSourceEffects are an

explosion with fixed location which increases and decreases radius, or a fireball with a fixed radius which moves from the front to the back of the room.

5.4. LightIteratorEffect

LightIteratorEffect will iterate an animation over *individual* lights with a certain offset, order and mode. The order indicates how lights are sorted (e.g. 'from front to back' or 'counter clockwise'). The mode indicates what happens after every light is iterated over: stop ('single'), repeat ('cycle') or revert ('bounce'). The offset indicates how much later the animation on the next light starts after the previous light. The LightIteratorEffect is clearly different from the previous effect type primitives because it iterates over individual light sources. This means that the total duration or the speed of an iteration over all lights depends on the number of lights in the setup. However, note that all effects type primitives discussed in this document share the property that a single effect definition works on every light setup.

Figure 7 shows an example, in which a creator wants to create a sequence of lighting up lights red one by one for one second, in a left to right order. In that case, he creates a LightIteratorEffect and specifies a color animation (stable red color for one second), order 'left to right', mode 'cycle' and step duration of one second.

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

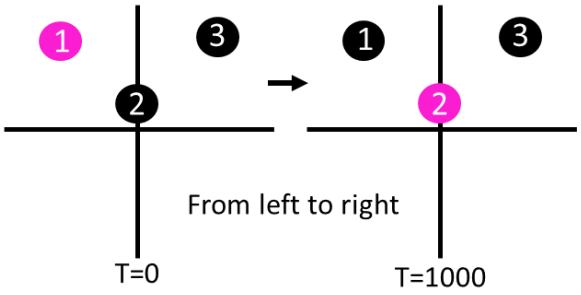


FIGURE 7: EXAMPLE LIGHTITERATOREFFECT MAPPING IN A POSSIBLE LIGHT SETUP

An example usage of a LightIteratorEffect is to build a chaser effect.

6. Animation type primitives

In above effect examples, we used mainly 'constant' animations for ease of explanation. But in general, every property of an effect can be animated by any type of animation. Just some examples:

- The opacity of an AreaEffect changes from 100% to 50% linearly over 500ms
- The color of a channel changes from red to blue in 2 seconds
- The radius of a LightSourceEffect changes from 0.5 to 1 quadratically over 5 seconds

Five animation primitive types are currently available and described below. More animation types could be added by deriving from the animation base class.

6.1.Constant

The constant animation models a value which is fixed over time.

```
Constant constant(0.25)
```

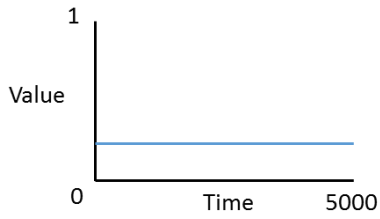


FIGURE 8: EXAMPLE OF A CONSTANT ANIMATION

6.2.Curve

A curve animation is a list of time-value points which are linearly interpolated.

```
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))
```

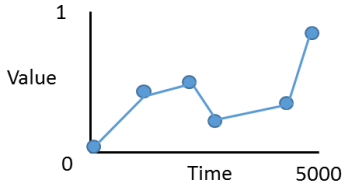


FIGURE 9: EXAMPLE OF A CURVE ANIMATION

6.3.Tween

A tween animation interpolates between a start and end point over a certain time via a certain trajectory such as linear, quadratic or sinusoidal.

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

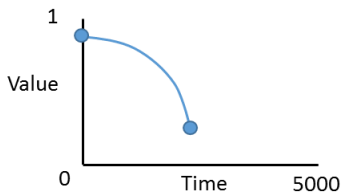


FIGURE 10: EXAMPLE OF A TWEEN ANIMATION

6.4.Sequence

A sequence animation is an ordered sequence of animations of any type which can be repeated multiple times. In the example in Figure 11, we create a sequence of the curve and tween automation defined

above, which is repeated once (i.e. played twice). And on top of that we create an overall sequence which appends a constant to the first created sequence.

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

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

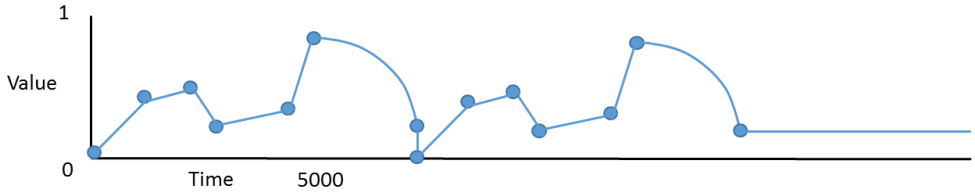


FIGURE 11: EXAMPLE OF A SEQUENCE ANIMATION

6.5. RandomTween

A 'RandomTween' animation generates random tweens between specified min/max interval and min/max values.

7. Combining effect with animation

Figure 12 depicts an example of using a LightSourceEffect with animations, to animate both the color, location and radius of the effect.

```
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))
```

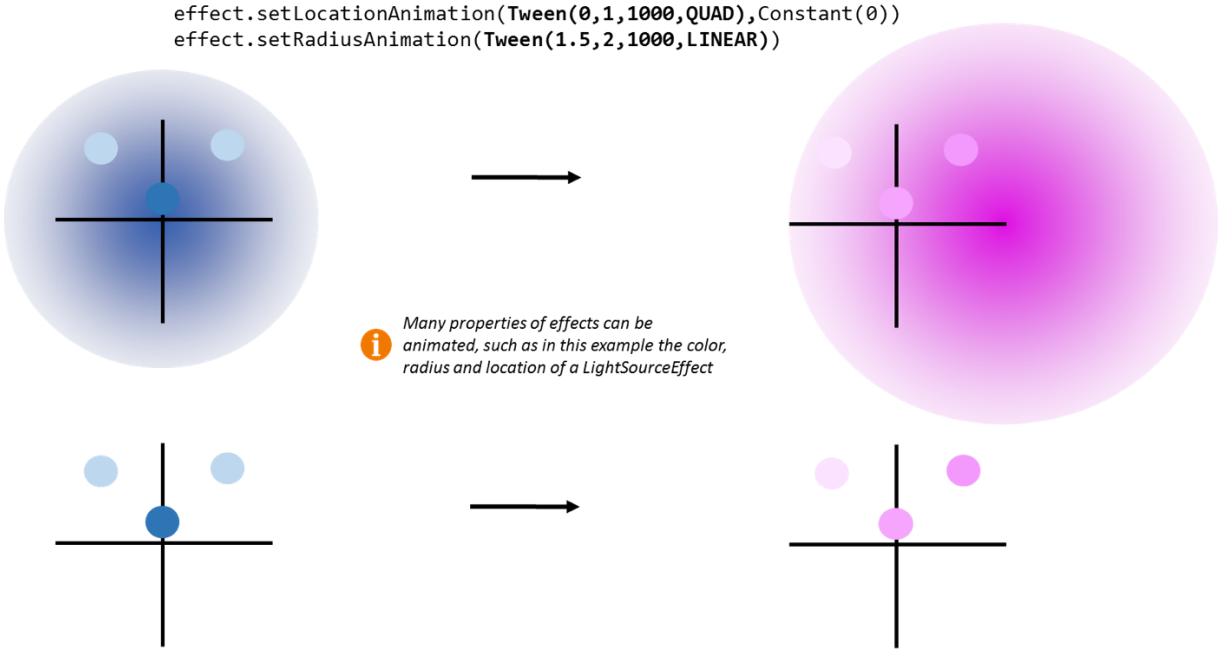


FIGURE 12: EXAMPLE OF USING ANIMATIONS IN A LIGHT EFFECT

8. Effect mixing

When multiple effects are enabled at the same time and affect the same lights, they will be mixed based on their layer (higher layers are rendered on top of lower layers) and opacity (the lower the opacity, the more the underlying effect will be blended in). An example of this is shown in Figure 13. Here, the blue effect resides on the top layer 1. Since it has an opacity of 0.65, it will be mixed for the top right light with the underlying red effect to form a purple color.

```
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))
```

FIGURE 13: EXAMPLE OF MIXING LIGHT EFFECTS

Effects are mixed using the simple 'Alpha blending' algorithm below.

```
Mixer::Render(Group entertainmentGroup) {
  //effects are sorted ascending on layer
  effects.each { effect.render() }
  entertainmentGroup.lights.each {
    mixedColor = 0,0,0
    effects.each {
      effectColor = effect.getColor(light)
      mixedColor = effectColor.rgb * effectColor.alpha + mixedColor * (1 – effectColor.alpha)
    }
    light.setColor(mixedColor)
  }
}
```

9. Lightscript

A lightscript can be used to create a predefined script of light effects which trigger at specific times.

A lightscript contains:

- Metadata
    - Name
    - Ideal setup
- The script itself
    - For earch layer a list of actions. Action is just a container for an animationEffect which
        - Adds a start position and optional explicit end position

- If an explicated end position is defined, this overrides the duration of underlying effect
- Takes care that the underlying effect runs on time provided by a timeline instead of ‘real’ time

A lightscript can be bound to a timeline. A timeline can play fully autonomous, or the time position can be fully controlled by the application, or a combination where it plays autonomously but is regularly synced by an application.

A lightscript can be imported and exported in JSON format. An example script with 4 actions can be generated with the EDK example application.

10. Overview

The class diagram in Figure 14 provides a summary overview of all concepts described in this document and how they relate to each other.

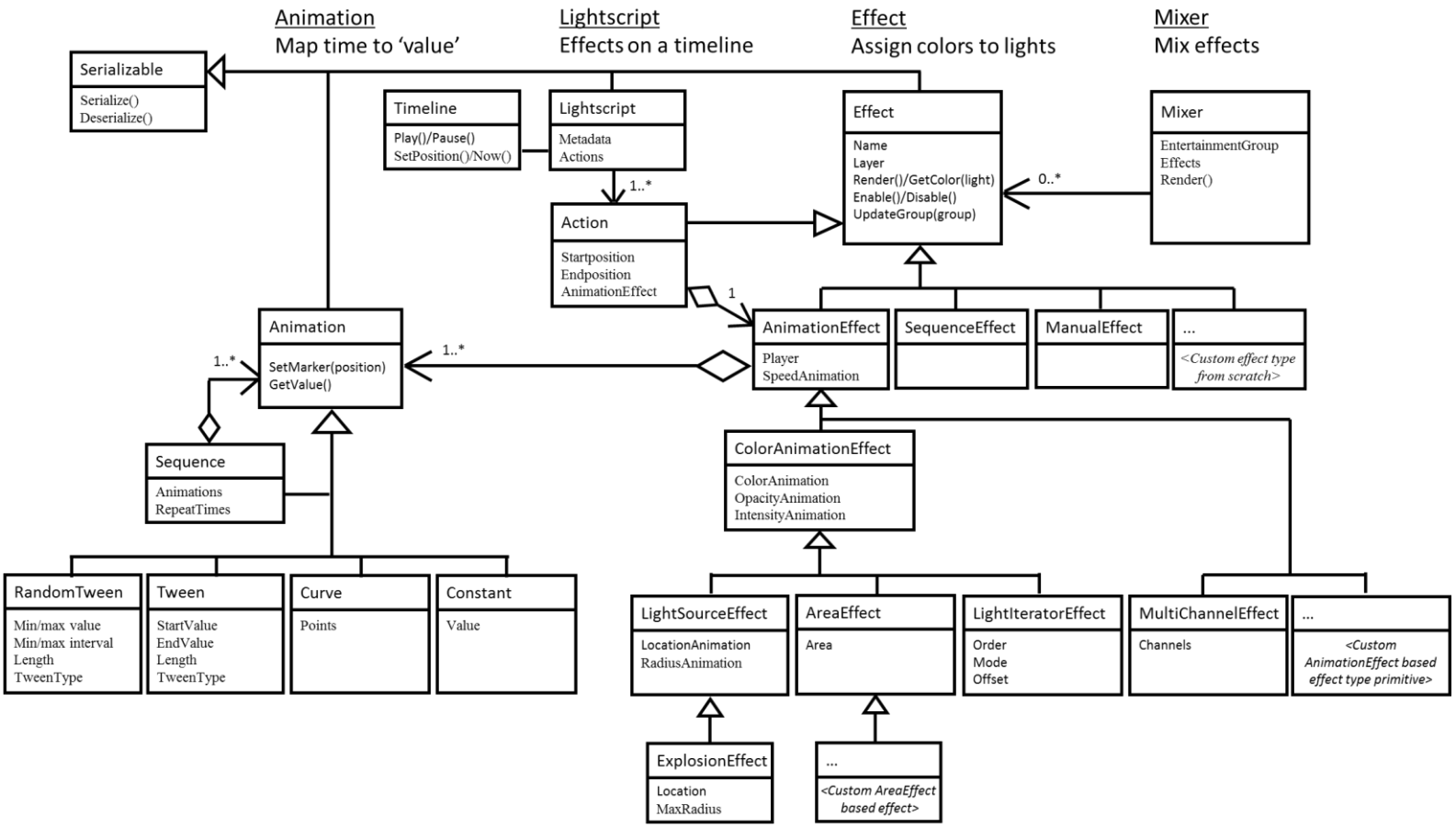


FIGURE 14: CLASS DIAGRAM OF HUE LIGHT EFFECT ENGINE

This overview shows the most relevant classes, a full class overview which includes every detail can be generated with DoxyGen based on the source code.

11. Further Expansion of Effects

Commonly used effects are pre-defined for:

- Creating ambient lighting
- Mimicking explosion
- Sparkles
- Damage taken
- Rumble visual effect
- Lightning
- Etc.

Developers can create additional effects based on the primitives described in this application note or create their own fully custom effect types.

An effect can be created from scratch by deriving from the effect base class. The most important method to implement in that case is `Color GetColor(Light light)`, where you have to assign a color for each light. Calculations which have do be done only once per render pass (rather than per light) can be implemented in the `void Render()` method. If information about the user's current entertainment setup is required, then the `void UpdateGroup(Group group)` method should be used.

If you want to build a new effect type which makes use of animations, then derive from AnimationEffect (or ColorAnimationEffect to include the standard color animations) instead of Effect. In this case `AnimationList GetAnimations()` should be implemented to provide the animations which should be animated.

Implement `Serialize(JSONNode node)` and `Deserialize(JSONNode node)` to be able to store and load effects from JSON text files. Source code of existing effect types can be used as examples.

12. Effect source code examples in C++

Complete source code of effect examples can be found at https://github.com/PhilipsHue/EDK

12.1. Example of applying effect: Explosion

Adding an ExplosionEffect to an application can be done easily as following code.

```
void ExplosionMenu::PlayExplosion(double x, double y) {
        auto explosion = std::make_shared<ExplosionEffect>("BAM!", 0);
        auto color = Color(1, 0.8, 0.4);
        auto radius = 0.5;
        auto intensity_expansion_time = 50;
        auto radius_expansion_time = 106;
        auto duration = 2000;
        explosion->PrepareEffect(color, huestream::Location(x, y), duration, radius,
radius_expansion_time, intensity_expansion_time);
        _huestream->LockEngine();
        _huestream->AddEffect(explosion);
        explosion->Enable();
        _huestream->UnlockEngine();
}
```

Other effects can be added in similar way. Create an instance, provide arguments, and add it to HueStream instance.

12.2. Example of applying effect with animation: SequenceAnimation in MultiChannelEffect

Adding a SequenceAnimation to an application also can be done easily as following code.

```
void EffectPlayer::PlayRedWhiteBlue() {
    auto animation = make_shared<SequenceAnimation>(0);
    auto slowUp = make_shared<TweenAnimation>(0, 1, 2500, TweenType::Linear);
    auto fastDown = make_shared<TweenAnimation>(1, 0.5, 500, TweenType::Linear);
    auto fastUp = make_shared<TweenAnimation>(0.5, 1, 500, TweenType::Linear);
    auto fastDownUpRepeated = make_shared<SequenceAnimation>(3);
    fastDownUpRepeated->Append(fastDown, "FastDown");
    fastDownUpRepeated->Append(fastUp, "FastUp");
    auto slowDown = make_shared<TweenAnimation>(1, 0, 2500, TweenType::Linear);
    animation->Append(slowUp, "SlowUp");
    animation->Append(fastDownUpRepeated, "FastDownUp");
    animation->Append(slowDown, "SlowDown");

    auto zero = make_shared<ConstantAnimation>(0);
    auto one = make_shared<ConstantAnimation>(1);

    auto redChannel = make_shared<Channel>(Location(-(2.0 / 3.0), 0), animation, zero, zero, one);
    auto whiteChannel = make_shared<Channel>(Location(0, 0), animation, animation, animation, one);
    auto blueChannel = make_shared<Channel>(Location((2.0 / 3.0), 0), zero, zero, animation, one);

    auto rwbEffect = std::make_shared<MultiChannelEffect>("rwb", 0);
    rwbEffect->AddChannel(redChannel);
    rwbEffect->AddChannel(whiteChannel);
    rwbEffect->AddChannel(blueChannel);
    rwbEffect->SetSpeedAnimation(m_speedAnimation);

    m_hueStream->LockMixer();
    m_hueStream->AddEffect(rwbEffect);
    rwbEffect->Enable();
    m_hueStream->UnlockMixer();
}
```

All other animations can be added in similar way. Create an instance, provide arguments, and add it to HueStream instance.

12.3. Example of applying derived effect: LightSourceEffect

Adding a derived effect to an application can be done as following code. In this code, RGBSineEffect is derived from LightSourceEffect.

```
class RGBSineEffect : public LightSourceEffect {
public:
  RGBSineEffect(const std::string &name, unsigned int layer);

  void PrepareEffect(std::shared_ptr<ConstantAnimation> speedAnimation,
                     std::shared_ptr<ConstantAnimation> alphaAnimation);
};

void EffectPlayer::SetSineEnabled(bool enabled) {
  if (enabled) {
    m_hueStream->LockMixer();
    if (m_sine == nullptr) {
      m_sine = std::make_shared<RGBSineEffect>("Sine", 6);
      m_sine->PrepareEffect(m_speedAnimation, m_alpha_sine);
      m_hueStream->AddEffect(m_sine);
    }
    m_sine->Enable();
    m_hueStream->UnlockMixer();
  } else if (m_sine != nullptr) {
    m_hueStream->LockMixer();
    m_sine->Disable();
    m_hueStream->UnlockMixer();
  }
}
```

12.4. Example of combining different effects

Following code shows how to combine multiple different effects, which are AreaEffect and LightSourceEffect.

```
void EffectPlayer::PlayCombi() {
```

```
    auto bcEffect = std::make_shared<AreaEffect>("Saw", 0);

    auto b = make_shared<SequenceAnimation>(4);
    b->Append(make_shared<TweenAnimation>(0, 1, 1000, TweenType::Linear), "Up");
    b->Append(make_shared<TweenAnimation>(1, 0, 1000, TweenType::Linear), "Down");

    auto g = make_shared<ConstantAnimation>(0);
    auto r = make_shared<ConstantAnimation>(0);

    bcEffect->SetColorAnimation(r, g, b);
    bcEffect->AddArea(Area::All);
    bcEffect->SetSpeedAnimation(m_speedAnimation);

    auto lsEffect = std::make_shared<LightSourceEffect>("RandomFlash", 1);
    auto length = 8000;
    auto i = make_shared<RandomAnimation>(0.1, 1, 200, 400, TweenType::Linear, length);

    lsEffect->SetColorAnimation(i, i, i);
    lsEffect->SetRadiusAnimation(i);

    auto x = make_shared<RandomAnimation>(-1, 1, 500, 1000, TweenType::Linear, length);
    auto y = make_shared<RandomAnimation>(-1, 1, 500, 1000, TweenType::Linear, length);
    lsEffect->SetPositionAnimation(x, y);
    lsEffect->SetOpacityAnimation(make_shared<ConstantAnimation>(0.8));
    lsEffect->SetSpeedAnimation(m_speedAnimation);

    m_hueStream->LockMixer();
    m_hueStream->AddEffect(bcEffect);
    m_hueStream->AddEffect(lsEffect);
    bcEffect->Enable();
    lsEffect->Enable();
    m_hueStream->UnlockMixer();
}
```