Table of Contents

---

1. Overview

`RCC_Light` is a **vehicle lighting script** within the **Realistic Car Controller (RCC)** framework. It manages **intensity** and **states** for various car lights:

- **Headlights** (low/high beams)
- **Brake** and **Reverse** lights
- **Indicators** (turn signals)
- **Parking**, **Interior**, and **External** lights

It also handles optional features like **lens flare**, **trail renderers** (e.g., neon undercarriage glow), **emission-based lighting** (mesh or texture emission), and **light damage / breakage**.

---

2. Class Declaration

[RequireComponent(typeof(Light))]
public class RCC_Light : RCC_Core {
  // ...
}

- Inherits from `RCC_Core`, which provides access to the parent **`RCC_CarControllerV4`** (via `CarController`), RCC Settings, etc.
- Requires a Unity **`Light`** component on the same GameObject.

---

3. Components and References

1. **`Light LightSource`**
   - Wraps a **`Light`** component (lazily loaded).
   - Encapsulates properties like `intensity`, `range`, `spotAngle`, and `renderMode`.
2. **`LensFlare lensFlare`**
   - Optional standard pipeline lens flare component.
3. **`LensFlareComponentSRP lensFlareURP`** *(only if `RCC_URP` is defined)*
   - URP-based lens flare component.
4. **`TrailRenderer trail`**
   - Optionally assigned for certain lights, e.g., neon glow effect.
5. **`RCC_Emission[] emission`**
   - An array of **emission** settings, used to manipulate textures or materials for extra glow or brightness effect.

---

4. Configuration Fields

- **`float defaultIntensity = 1f`**
  The default on-state intensity of the light.

- **`float flareBrightness = 1.5f`**
  Maximum brightness factor for lens flare. Adjusted over distance and angle.

- **`float inertia = 1f`**
  Smoothness factor for transitioning the light’s intensity (bigger means slower).

- **`LightRenderMode renderMode = LightRenderMode.Auto`**
  Defines whether the light is vertex-based, pixel-based, or auto.

- **`bool overrideRenderMode = false`**
  If `true`, user sets `renderMode` manually. If `false`, RCC automatically decides based on `lightType`.

- **`Flare flare`**
  The lens flare asset, if any, used when `lensFlare` is present.

- **`int refreshRate = 30`**
  Lens flare calculation frequency (in FPS). Lower rates can improve performance but reduce accuracy.

- **`bool useEmissionTexture = false`**
  If `true`, uses mesh emission to visually enhance the light.

4.1 LightType Enum

```
public enum LightType {
    HeadLight,
    BrakeLight,
    ReverseLight,
    Indicator,
    ParkLight,
    HighBeamHeadLight,
    External,
    Interior
}
```

Specifies the **role** of the light:

- **`HeadLight`**: Standard headlight (low beam).
- **`BrakeLight`**: Illuminated when braking.
- **`ReverseLight`**: Illuminated in reverse gear.
- **`Indicator`**: Turn signals/hazards.
- **`ParkLight`**: Parking / tail light.
- **`HighBeamHeadLight`**: Separate high beam logic.
- **`External`**: Additional external lights not automatically toggled.
- **`Interior`**: Cabin illumination, toggled independently.

4.2 Breaking / Damage Fields

- **`bool isBreakable = true`**
  If `true`, collisions can reduce `strength` and eventually break the light.

- **`float strength = 100f`**
  Current structural integrity of the light. Collisions reduce this.

- **`int breakPoint = 35`**
  Threshold below which the light becomes **broken** (no longer emits light).

- **`private bool broken = false`**
  Tracks if the light is currently broken.

4.3 Emission

- **`RCC_Emission[] emission`**
  Each array element can manipulate material properties to simulate glowing or bright surfaces, especially for brake lights or headlights.

---

5. Initialization and Lifecycle

5.1 `Awake()`

```
private void Awake() {
    Initialize();
}
```

- Ensures the `Initialize()` method is called once this component is awakened.

5.2 `Initialize()`

1. Loads references to `lensFlare`, `lensFlareURP`, and `trail`.
2. Ensures the **`Light`** is **enabled** and sets `defaultIntensity` if not already assigned.
3. Applies initial lens flare settings if present.
4. Determines **render mode** based on user overrides or RCC settings (e.g., `useHeadLightsAsVertexLights`).
5. If attached to an RCC vehicle, finds or creates audio sources (for indicator beep) and checks for sibling lights to detect **parkLightFound** or **highBeamLightFound**.

5.3 `OnEnable()`

private void OnEnable() {
    LightSource.intensity = 0f;
}

- Ensures the light is turned off initially upon enabling.

5.4 `Update()`

1. If no `CarController`, return.
2. Calls **`LensFlare()`** and **`TrailRenderer()`** if available.
3. Applies **emission** updates if `useEmissionTexture` is `true`.
4. If **broken**, sets intensity to `0f` and returns immediately.
5. Otherwise, checks the **`lightType`** and updates intensity based on **vehicle state** (e.g., brakes engaged, reverse gear, turn signals).

---

6. Lighting Logic

Core method:

private void Lighting(float input)
private void Lighting(float input, float range, float spotAngle)

- Smoothly interpolates the **`LightSource.intensity`** toward `input` using `Mathf.Lerp`, multiplied by `Time.deltaTime * inertia * 20f`.
- Overloaded variant also sets `range` and `spotAngle`.

6.1 Headlights, Brake Lights, Reverse, etc.

When `lightType` is:

- **`HeadLight`**
  - If `highBeamLightFound`, only lit when **low beam** is on.
  - Else checks if **lowBeamHeadLightsOn** / **highBeamHeadLightsOn** to decide intensity.
- **`BrakeLight`**
  - If `parkLightFound`, fully on only when brakeInput >= 0.1.
  - Else can partially light when headlights are on, but full intensity on brake.
- **`ReverseLight`**
  - On if `CarController.direction == -1`.
- **`ParkLight`**
  - On if `CarController.lowBeamHeadLightsOn`.

6.2 Indicators

- **`Indicator`**
  - Determines left/right side by comparing local X position to the vehicle's transform.
  - Uses a timer (`CarController.indicatorTimer`) for a blink cycle.
  - **Indicator Audio**: Plays once at the start of each "on" cycle.

6.3 High Beam

- **`HighBeamHeadLight`**
  - Intensity only if `CarController.highBeamHeadLightsOn`.
  - Usually has a higher range and narrower spot angle.

6.4 Interior Lights

- **`Interior`**
  - On if `CarController.interiorLightsOn`.
  - Typically unaffected by external factors like braking or reversing.

6.5 External Lights

- **`External`**
  - Script does not automatically manage them; you need custom code to set their intensity if desired.

---

7. Lens Flare Calculation

```
private void LensFlare() {
   // Only updates every (1 / refreshRate) seconds to improve performance
   // ...
   // finalFlareBrightness = flareBrightness * (distance, angle factors) ...
   // lensFlare.brightness = finalFlareBrightness * LightSource.intensity
}
```

- If `lensFlare` or `lensFlareURP` is present, calculates brightness based on:
  - **Distance** to `Camera.main`.
  - **Angle** between light forward vector and camera direction.
  - Applied each cycle or at a reduced frequency (`refreshRate`) to optimize performance.

---

8. Trail Renderer Handling

```
private void TrailRenderer() {
    trail.emitting = LightSource.intensity > .1f;
    trail.startColor = LightSource.color;
}
```

- Simple logic: if the light intensity > 0.1, the trail “emits” (visible).
- Matches the color to the current `LightSource.color`.

---

9. Damage and Repair Methods

1. **`public void OnCollision(float impulse)`**
    - Decreases `strength` by `impulse * 20f`.
    - If `strength` <= `breakPoint`, sets `broken = true`.
2. **`public void OnRepair()`**
    - Restores `strength` to its original value (`orgStrength`).
    - Sets `broken = false`, re-enabling illumination.

---

10. Usage Notes and Best Practices

1. **One `RCC_Light` Per Light**
    - Attach the script to each **Light** component that needs RCC logic (brakes, indicators, etc.).
2. **Check Prefab Orientation**
    - The `Reset()` method ensures correct orientation of headlights/tail-lights relative to the vehicle’s forward direction.
3. **Breaking**
    - Only relevant if `isBreakable = true`. Collisions invoked from `RCC_Damage` or manual calls to `OnCollision(float)`.
4. **Performance**
    - The `refreshRate` for lens flare can be lowered in large scenes or on mobile to save CPU time.
    - Automatic vertex vs. pixel light assignment can also help optimize performance.
5. **Indicators**
    - The script uses a shared `indicatorSound` on the vehicle; only plays once each blink cycle.

    - If you need custom sounds, you can replace or modify the `IndicatorClip` reference in `RCC_Settings`.
6. **Compatibility**
    - For URP lens flares, `lensFlareURP` is used if `RCC_URP` is defined. Otherwise, standard pipeline fallback.
7. **Custom Light Behavior**
    - For `External` lights, manually set intensity from your own scripts or UI toggles.

---

11. Summary

`RCC_Light` provides a **comprehensive system** for **car lights** in Realistic Car Controller. From **headlight beams** and **brake lights** to **indicators** and **interior illumination**, it automatically adjusts intensities based on the vehicle's state. Additionally, it can **break** via collisions, features **lens flare** logic, supports **trail renderers**, and integrates **mesh emission** for a realistic, high-quality lighting setup within RCC.