Table of Contents

1. Overview
2. Class Declaration
3. Purpose and Functionality
4. Vehicle Instantiation and Registration
   4.1 Spawning Vehicles
   4.2 Registering and De-Registering as Player Vehicle
5. Vehicle States and Controls
   5.1 Controlling CanControl
   5.2 Starting / Killing Engine
6. Mobile and Behavior Settings
7. Record and Replay Methods
8. Camera Management
9. Transporting Vehicles
10. Skidmarks Management
11. Repair Methods
12. Usage Notes and Best Practices
13. Summary

---

1. Overview

`RCC` is a **static utility class** for **Realistic Car Controller (RCC)**. It offers **quick-access** methods to spawn new vehicles, register or de-register them as player vehicles, start/kill engines, modify behaviors (arcade vs. realistic), initiate recording or replay, etc. By centralizing these calls, developers can easily manage RCC vehicles at runtime without directly referencing internal `RCC_SceneManager` or `RCC_Settings` details.

---

2. Class Declaration

```
public class RCC {
    // ...
}
```

- Not a `MonoBehaviour`; it is purely **static** methods.

---

3. Purpose and Functionality

1. **Spawn** an RCC vehicle at a specific location and orientation.
2. **Register** a vehicle as the player's active vehicle or set it as **controllable**.
3. **Engine** management (start/kill).
4. **Mobile** controller and **driving behavior** toggles.
5. **Record / Replay** session handling.
6. **Transport** vehicles (teleport them).
7. **Clean** or remove skidmarks.
8. **Repair** damage on a vehicle.

These methods delegate the real work to **`RCC_SceneManager`**, **`RCC_SkidmarksManager`**, or directly on the `RCC_CarControllerV4`.

---

4. Vehicle Instantiation and Registration

4.1 Spawning Vehicles

```
public static RCC_CarControllerV4 SpawnRCC(
    RCC_CarControllerV4 vehiclePrefab,
    Vector3 position,
    Quaternion rotation,
    bool registerAsPlayerVehicle,
    bool isControllable,
    bool isEngineRunning
)
```

- **`vehiclePrefab`**: The RCC car prefab to spawn.
- **`position`**, **`rotation`**: Where to instantiate it.
- **`registerAsPlayerVehicle`**: If `true`, calls `RCC_SceneManager.RegisterPlayer()` for the newly created vehicle.
- **`isControllable`**: Whether to let the vehicle respond to input.
- **`isEngineRunning`**: If `true`, immediately starts the engine; if `false`, the vehicle spawns with a killed engine.
- Returns the **`RCC_CarControllerV4`** instance.

```
// Example:
RCC_CarControllerV4 myCar = RCC.SpawnRCC(myPrefab, spawnPos, spawnRot, true, true, true);
```

4.2 Registering and De-Registering as Player Vehicle

```
public static void RegisterPlayerVehicle(RCC_CarControllerV4 vehicle) { ... }
```

public static void RegisterPlayerVehicle(RCC_CarControllerV4 vehicle, bool isControllable) { ... }
public static void RegisterPlayerVehicle(RCC_CarControllerV4 vehicle, bool isControllable, bool engineState) { ... }
public static void DeRegisterPlayerVehicle() { ... }

- Allows different overloads for setting controllability and engine state:
  - `RegisterPlayerVehicle(vehicle, true, true)` => Vehicle is player, controllable, engine on.
- `DeRegisterPlayerVehicle()` => Clears the current player vehicle from `RCC_SceneManager`.

---

5. Vehicle States and Controls

5.1 Controlling `CanControl`

public static void SetControl(RCC_CarControllerV4 vehicle, bool isControllable);

- Directly toggles a specific vehicle's `.SetCanControl(isControllable)`.

5.2 Starting / Killing Engine

public static void SetEngine(RCC_CarControllerV4 vehicle, bool engineState);

- If `engineState` is true => `vehicle.StartEngine()`, else => `vehicle.KillEngine()`.

---

6. Mobile and Behavior Settings

public static void SetMobileController(RCC_Settings.MobileController mobileController);
public static void SetBehavior(int behaviorIndex);

- **`SetMobileController`**: Updates `RCC_Settings.Instance.mobileController`.
- **`SetBehavior`**: Calls `RCC_SceneManager.Instance.SetBehavior(behaviorIndex)`, switching the handling style globally.

---

7. Record and Replay Methods

public static void StartStopRecord();
public static void StartStopReplay();
public static void StopRecordReplay();

- **`StartStopRecord()`** toggles between *neutral* and *record* modes in `RCC_SceneManager`.
- **`StartStopReplay()`** toggles between *neutral* and *play* modes.
- **`StopRecordReplay()`** forces a stop from either record or play.

---

8. Camera Management

public static void ChangeCamera();

- Calls `RCC_SceneManager.Instance.ChangeCamera()`, cycling through camera modes (hood, orbit, chase, etc.).

---

9. Transporting Vehicles

public static void Transport(Vector3 position, Quaternion rotation);
public static void Transport(RCC_CarControllerV4 vehicle, Vector3 position, Quaternion rotation);

- Teleports either:
  - The **active** player vehicle to `position, rotation`, or
  - A **specific** `vehicle` reference to `position, rotation`.
- Internally calls `RCC_SceneManager` teleport methods, resetting velocity, applying a short freeze.

---

10. Skidmarks Management

public static void CleanSkidmarks();
public static void CleanSkidmarks(int index);

- Invokes `RCC_SkidmarksManager.Instance.CleanSkidmarks()`, clearing all skidmarks or just the ones from a specific vehicle index if the manager supports that filtering.

---

11. Repair Methods

public static void Repair(RCC_CarControllerV4 carController);
public static void Repair();

- **`Repair(RCC_CarControllerV4 carController)`** => sets `carController.damage.repairNow = true;`
- **`Repair()`** => repairs the **active player** vehicle if one exists.
- This triggers the vehicle's damage script to gradually restore to an undamaged state.

---

12. Usage Notes and Best Practices

1. **Convenience Calls**
    - Most methods are wrappers around `RCC_SceneManager` or direct calls on the vehicle's `RCC_CarControllerV4` instance.
    - Great for hooking up **UI buttons** or **quick debugging**.
2. **Scene Manager**
    - Ensure you have **`RCC_SceneManager`** in the scene. The `RCC` static calls rely on it being present and valid.
3. **Spawning**
    - `SpawnRCC(...)` is the recommended way to instantiate new RCC vehicles at runtime. You can still manually `Instantiate(prefab)`, but using the `RCC` method ensures correct setup.
4. **Recording**
    - Make sure your vehicle(s) have **`RCC_Recorder`** if you want to use record/replay.
    - The `StartStopRecord()` and `StartStopReplay()` methods toggle the *global* record/replay states in the scene manager.
5. **Behavior**
    - `SetBehavior(int index)` changes the driving style (arcade, semi-arcade, realistic, etc.) for **all vehicles**, as set in `RCC_Settings.behaviorTypes`.
6. **Multiple Vehicles**
    - If multiple vehicles are spawned, `RegisterPlayerVehicle(...)` decides which one is the *active player* for controls and camera.

7. **Mobile**
    - `SetMobileController(...)` can instantly switch the UI layout (touchscreen, gyro, steering wheel, joystick) if `mobileControllerEnabled` is true in `RCC_Settings`.

---

13. Summary

`RCC` is a high-level **static API** for **Realistic Car Controller**, simplifying:

- **Spawning** vehicles
- **Registering** them as player vehicles
- **Controlling** engines and **inputs**
- **Recording** or **Replaying** sessions
- **Camera** toggling
- **Teleporting** (transport)
- **Skidmark** cleanup
- **Repair** states

All with straightforward, single-line method calls that under-the-hood delegate tasks to `RCC_SceneManager`, `RCC_Settings`, or other RCC subsystems.