Table of Contents

1. Overview
2. Class Declaration
3. Purpose and Key Features
4. Core Fields and Properties
    4.1 Control State Flags
    4.2 Wheels
    4.3 Steering Wheel Model
    4.4 Drivetrain Setup
    4.5 AI and External Control
    4.6 Steering Configuration
    4.7 Vehicle Configuration and Rigidbody
    4.8 Engine / RPM Setup
    4.9 Steering Assistance Features
    4.10 Fuel System
    4.11 Engine Heat System
    4.12 Gears and Transmission
    4.13 Audio System
    4.14 Vehicle Inputs
    4.15 Lights (Headlights, Indicators, Interior)
    4.16 Damage Handling
    4.17 Driver Assistances (ABS, TCS, ESP)
    4.18 Drift Logic
    4.19 Turbo / NOS
5. Events
6. Workflow / Lifecycle Methods
    6.1 Awake()
    6.2 OnEnable()
    6.3 Update()
    6.4 FixedUpdate()
    6.5 OnDisable()
    6.6 OnDestroy()
7. Core Functions
    7.1 Engine, Gears, and Clutch
    7.2 Input and Steering Logic
    7.3 Wheel Force Application
    7.4 Driving Assistances
    7.5 Audio Handling
    7.6 Damage and Collision Response
    7.7 Vehicle Reset Check
8. Overriding Inputs
9. Trailer and Preview Modes
10. Editor / Debug Utilities

11. Usage Notes and Best Practices
12. Summary

---

1. Overview

`RCC_CarControllerV4` is the **main vehicle controller** script for the Realistic Car Controller (RCC). It provides comprehensive car simulation features, including:

- Wheel colliders setup
- Steering, drivetrain, and gear logic
- Engine torque, RPM, clutch, brake, and fuel consumption
- Vehicle stability, driver assists (ABS, TCS, ESP)
- Visual effects (lights, particles, skidmarks), audio sources (engine, turbo, NOS)
- Collision-based damage with mesh deformation
- Drifting logic and advanced steering limitations

All high-level RCC systems rely on this script to simulate realistic driving behavior.

---

2. Class Declaration

```
[RequireComponent(typeof(Rigidbody))]
public class RCC_CarControllerV4 : RCC_Core {
    // Implementation...
}
```

- Inherits from `RCC_Core`, granting access to RCC Settings, audio helper methods, etc.
- Requires a `Rigidbody` component on the same GameObject.

---

3. Purpose and Key Features

- **CanControl**: Toggles standard input from RCC_InputManager.
- **Wheel System**: Configures front/rear/extra wheels, powers them according to drivetrain type.
- **Engine**: Tracks RPM, torque curves, rev limit, engine state (running/stopped).
- **Transmission**: Gears, automatic/semi-auto/manual modes, gear shifting logic.
- **Steering**: Steering angle curves for high-speed limiting, drift/counter-steer modes.
- **Assist Systems**: ABS, TCS, ESP, traction helper, steering helper.
- **Damage**: Collision-based mesh deformation, spark particles, crash audio.
- **NOS/Turbo**: Adds short boost or spool effects.

- **Audio**: Multiple engine layers (idle, low, mid, high, off versions), brake squeal, wind noise.
- **Lights**: Headlights, indicators, interior, brake, reverse.
- **Fuel/Heat**: Optionally track engine temperature and fuel usage.

---

4. Core Fields and Properties

4.1 Control State Flags

- `public bool canControl`: If false, vehicle ignores standard user inputs.
- `public bool isGrounded`: True if any wheel is grounded.
- `public bool overrideBehavior`: If true, does not apply RCC_Settings selectedBehaviorType on this vehicle.
- `public bool overrideInputs`: If true, uses `OverrideInputs()` instead of normal RCC_InputManager.

4.2 Wheels

- `Transform FrontLeftWheelTransform, FrontRightWheelTransform, RearLeftWheelTransform, RearRightWheelTransform`
  The visual wheel meshes for the standard 4 wheels.
- `Transform[] ExtraRearWheelsTransform` for vehicles with >4 wheels.
- `RCC_WheelCollider FrontLeftWheelCollider, ...` references the associated physics wheel colliders.
- `bool hasExtraWheels`: For vehicles like trucks with multiple rear axles.
- `bool overrideAllWheels`: If true, standard power/steer/brake settings apply across all wheels.
- `int poweredWheels`: The count of wheels that actually receive power (torque).

4.3 Steering Wheel Model

- `public Transform SteeringWheel`: The 3D interior steering wheel.
- `public enum SteeringWheelRotateAround { XAxis, YAxis, ZAxis }` and `steeringWheelRotateAround`: Defines the axis of rotation.
- `public float steeringWheelAngleMultiplier = 11f`: How many degrees the 3D wheel rotates relative to actual steering angle.

4.4 Drivetrain Setup

- `public enum WheelType { FWD, RWD, AWD, BIASED }` and `wheelTypeChoise`: Drivetrain style (front/rear/all-wheel drive).

4.5 AI and External Control

- `public bool externalController`: If true, **no standard input** is applied. Throttle/brake/steer can be set externally (e.g. AI).

4.6 Steering Configuration

- `public enum SteeringType { Curve, Simple, Constant }` and `steeringType`
- `public AnimationCurve steerAngleCurve` for advanced high-speed limiting.
- `public float steerAngle`: The maximum angle at low speed.
- `public float highspeedsteerAngle, highspeedsteerAngleAtspeed`: Limits angle at higher speeds.
- `public float antiRollFrontHorizontal, antiRollRearHorizontal, antiRollVertical`: Anti-roll bar strengths.

4.7 Vehicle Configuration and Rigidbody

- `public Rigidbody Rigid { get; }`: Cached reference to the `Rigidbody`.
- `public Transform COM`: Center of Mass object.
- `public enum COMAssisterTypes { Off, Slight, Medium, Opposite }`: Adjusts COM in real-time for leaning or stability.
- `public float brakeTorque`: Maximum brake torque.
- `public float downForce`: Aerodynamic downforce multiplied by speed.
- `public float speed`: Current speed in km/h.
- `public float maxspeed`: Computed top speed.
- `public bool limitMaxSpeed, float limitMaxSpeedAt`: Option to clamp max speed.
- `private float resetTime`: Time spent upside down to auto-reset.

4.8 Engine / RPM Setup

- `public AnimationCurve engineTorqueCurve`: Defines torque vs. RPM.
- `public float maxEngineTorque, maxEngineTorqueAtRPM`: Peak torque and at which RPM it occurs.
- `public float minEngineRPM, maxEngineRPM`: Idle and rev limit.
- `public float engineRPM, engineRPMRaw`: The current engine RPM (smoothed or raw).
- `public float engineInertia = 0.15f`: How quickly engine revs up or down.
- `public bool useRevLimiter, useExhaustFlame`: Additional features.
- `public bool RunEngineAtAwake`: Start engine automatically on spawn?
- `public bool engineRunning`: Current engine on/off state.

4.9 Steering Assistance Features

- `public bool useSteeringLimiter, useCounterSteering, useSteeringSensitivity`
  Various drift/counter-steer or angle-limit systems.
- `public float counterSteeringFactor`: Additional input for drifting.
- `public float steeringSensitivityFactor`: Damps or smooths out user steering input.

4.10 Fuel System

- `public bool useFuelConsumption`: If true, vehicle consumes fuel.
- `public float fuelTankCapacity, fuelTank, fuelConsumptionRate`: Basic consumption model.
- If **fuel** hits 0, engine shuts off.

4.11 Engine Heat System

- `public bool useEngineHeat`: If true, track engine temperature.
- `public float engineHeat`: Current temperature in °C.
- `public float engineCoolingWaterThreshold`: Temperature above which it cools.
- `public float engineHeatRate, engineCoolRate`: Warm-up and cool-down speeds.

4.12 Gears and Transmission

```
[System.Serializable]
public class Gear {
    public float maxRatio;
    public int maxSpeed;
    public int targetSpeedForNextGear;
    // ...
}
```

- **`public Gear[] gears`**: Array describing gear ratios and shift speeds.
- **`public int totalGears`**: Number of forward gears.
- **`public bool automaticGear, semiAutomaticGear, automaticClutch`**: Transmission modes.
- **`public int currentGear`**: Active gear index, 0-based.
- **`public bool NGear`**: If true, neutral gear.
- **`public float finalRatio`**: Final drive multiplier.
- **`public bool changingGear`**: Clutch/fuel cut in progress.
- **`public int direction`**: 1 = forward, -1 = reverse, 0 = neutral.

4.13 Audio System

- **`public enum AudioType { OneSource, TwoSource, ThreeSource, Off }`**: The layering of engine sounds.
- **Engine Audio Sources**: `engineSoundHigh/Med/Low/Idle` + corresponding off variations.
- **Additional**: reversingSound, windSound, brakeSound, NOSSound, turboSound, crashSound.
- **Pitch/Volume**: Ramped by engine RPM, throttle, or speed.

4.14 Vehicle Inputs

- **`public RCC_Inputs inputs`**: The main container for throttle, brake, steer, etc.
- Processed into final floats: `throttleInput, brakeInput, steerInput, handbrakeInput, boostInput, clutchInput`.
- **`public bool cutGas, permanentGas`**: Force throttle cut or force full gas.

4.15 Lights (Headlights, Indicators, Interior)

- **`public bool lowBeamHeadLightsOn, highBeamHeadLightsOn, interiorLightsOn`**

- **`public IndicatorsOn indicatorsOn`**: Off, Right, Left, or All (hazard).
- **`public float indicatorTimer`**: Blinks indicators.

4.16 Damage Handling

- **`public bool useDamage`**: If true, collisions can cause mesh deformation via `RCC_Damage`.
- **`public RCC_Damage damage`**: Reference to the damage system.
- **`public bool useCollisionParticles, useCollisionAudio`**: Sparks and collision sounds.
- Pools spark particle systems to avoid constant instantiations.

4.17 Driver Assistances (ABS, TCS, ESP)

- **`public bool ABS, TCS, ESP`**: Toggles for brake lock prevention, traction control, stability program.
- **`public bool steeringHelper, tractionHelper, angularDragHelper`**: Additional stabilizers.
- **`public float ABSThreshold, TCSStrength, ESPThreshold, ESPStrength, ...`**: Tuning parameters.
- **`public bool ABSAct, TCSAct, ESPAct`**: Active states in the current frame.
- **`public float frontSlip, rearSlip`**: Slip angles used to detect understeer/oversteer.

4.18 Drift Logic

- **`public bool driftMode`**: If true, fosters drifting behavior.
- **`internal bool driftingNow`**: If rear slip angle is above a threshold.
- **`internal float driftAngle`**: The magnitude of the drift slip.

4.19 Turbo / NOS

- **`public bool useNOS, useTurbo`**: Toggles for nitrous or turbo spool.
- **`public float turboBoost`**: Ranges 0–30, used for spool audio.
- **`public float NoS`**: NOS level (0–100). Decreases during usage and regenerates.

---

5. Events

public delegate void onRCCPlayerSpawned(RCC_CarControllerV4 RCC);
public static event onRCCPlayerSpawned OnRCCPlayerSpawned;

public delegate void onRCCPlayerDestroyed(RCC_CarControllerV4 RCC);
public static event onRCCPlayerDestroyed OnRCCPlayerDestroyed;

public delegate void onRCCPlayerCollision(RCC_CarControllerV4 RCC, Collision collision);
public static event onRCCPlayerCollision OnRCCPlayerCollision;

- `OnRCCPlayerSpawned` fired when a *player-controlled* vehicle is enabled.
- `OnRCCPlayerDestroyed` when the vehicle is destroyed or disabled.
- `OnRCCPlayerCollision` fired on collisions if the vehicle is the active player vehicle.

---

6. Workflow / Lifecycle Methods

6.1 Awake()

- Sets up `Rigidbody` constraints, default gear/engine parameters.
- Assigns each wheel transform to the matching `RCC_WheelCollider`.
- Optionally starts the engine if `RunEngineAtAwake` is true.
- Creates necessary audio sources (engine, reversing, wind, brake, etc.).
- If `overrideBehavior` is false, calls `CheckBehavior()` to apply RCC_Settings.

6.2 OnEnable()

- Resets gear states, drifting flags, and input.
- Triggers a delayed **OnRCCPlayerSpawned** event if not externally controlled.
- Subscribes to `RCC_SceneManager.OnBehaviorChanged` to refresh behavior on global changes.
- Subscribes to `RCC_InputManager` events (headlights, indicators, gear shifts, etc.).

6.3 Update()

1. **Inputs()**: Gets inputs from `RCC_InputManager` or external override.
2. **Audio()**: Updates brake/wind or other constant audio volumes.
3. **CheckReset()**: Auto-reset if upside down for too long.
4. **Damage**: If `useDamage` is true, calls `damage.UpdateRepair()` and `damage.UpdateDamage()`.
5. **OtherVisuals()**: Rotates the interior steering wheel model.

6.4 FixedUpdate()

1. **CalculateMaxSpeed()**: Figures out top speed and gear speed thresholds.
2. **COM** adjustments (if `COMAssister` is active).
3. **Engine()**: Recomputes engine RPM from gear ratio, wheel speed, throttle.
4. **Steering()**: Adjusts `steerAngle` based on speed or curves.
5. **Wheels()**: Applies torque, brake, steering force to each wheel collider.
6. **AutomaticClutch()** + **AutomaticGearbox()** if in auto/semi-auto mode.
7. **AntiRollBars()**: Minimizes body roll.
8. **CheckGrounded()**: Sets `isGrounded`.
9. **RevLimiter()**, **Turbo()**, **NOS()**, **Fuel()**, **EngineHeat()** if respective features are enabled.
10. **SteerHelper()**, **TractionHelper()**, **AngularDragHelper()** if toggled.
11. **ESPCheck()**: Sums slip angles, sets `underSteering`, `overSteering`, applies corrections.

6.5 OnDisable()

- Unsubscribes from behavior/input manager events.
- Leaves final states intact.

6.6 OnDestroy()

- Fires **OnRCCPlayerDestroyed** event if applicable.

---

7. Core Functions

7.1 Engine, Gears, and Clutch

- **`Engine()`**: Merges traction wheel RPM with engine inertia, factoring in throttle/clutch to yield final `engineRPM`.
- **`AutomaticGearbox()`**: Shifts up/down automatically if `automaticGear` is true, based on `gearShiftUpRPM`, `gearShiftDownRPM`, and speed.
- **`ChangeGear(int gear)`**: Coroutine that sets `changingGear = true`, cuts throttle, then after delay sets `currentGear`.

7.2 Input and Steering Logic

- **`Inputs()`**: Reads from `RCC_InputManager` or `OverrideInputs()` if `overrideInputs = true`. Applies smoothing, limiting, or counter-steer.
- **`Steering()`**: Adjusts `steerAngle` via `steeringType` (Curve, Simple, or Constant).

7.3 Wheel Force Application

- **`Wheels()`**: Loops through `AllWheelColliders` applying motor torque, brake torque, and steering. Divides torque among `poweredWheels`.

7.4 Driving Assistances

- **`SteerHelper()`**: Nudges angular velocity to align with forward direction if drifting slightly.
- **`TractionHelper()`**: Reduces sideways stiffness for front wheels if rotation is excessive.
- **`AngularDragHelper()`**: Increases `Rigidbody.angularDrag` proportionally to speed.
- **`ESPCheck()`**: Compares front and rear slip with `ESPThreshold`, toggles `underSteering` or `overSteering`.

7.5 Audio Handling

- **`Audio()`**: Adjusts wind/brake volumes. Calls `EngineSounds()`.
- **`EngineSounds()`**: Depending on `audioType` (One/Two/ThreeSource), adjusts volumes/pitches for idle, low, med, high, off layers by engine RPM/throttle.

7.6 Damage and Collision Response

- **`OnCollisionEnter/Stay/Exit(Collision collision)`**: Checks collision magnitude, triggers spark particles, calls `damage.OnCollision()` if `useDamage = true`. Plays crash audio if needed.

7.7 Vehicle Reset Check

- **`CheckReset()`**: If speed < 5 and the vehicle is upside down for more than 3 seconds, it repositions the car upright.

---

8. Overriding Inputs

public void OverrideInputs(RCC_Inputs newInputs);
public void DisableOverrideInputs();

- **`overrideInputs = true`** stops standard input and uses `newInputs` for throttle/brake/steering.
- **`DisableOverrideInputs()`** reverts to normal input flow.

---

9. Trailer and Preview Modes

- **`public RCC_TruckTrailer attachedTrailer`**: If assigned, the vehicle can detach the trailer via `DetachTrailer()`.
- **`PreviewSmokeParticle(bool state)`**: Sets vehicle to a kinematic, no-control state with permanent full throttle (for smoke/burnout preview in a showroom).

---

10. Editor / Debug Utilities

- **`CreateWheelColliders()`**: Primarily used in editor scripts to ensure proper wheel colliders.
- **`CheckCOMPosition()`**: Moves the `COM` object to the approximate center of bounding box.
- **`Reset()`**: Called by Unity's editor on re-adding the component, sets typical default RCC values.

---

11. Usage Notes and Best Practices

1. **Wheel Setup**
   - Assign each `FrontLeftWheelCollider` → `FrontLeftWheelTransform` carefully. Mismatches cause steering or rotation issues.
2. **Behavior**
   - If `overrideBehavior = false`, the script auto-applies `Settings.selectedBehaviorType` from `RCC_Settings`. This can drastically change braking, steering, TCS, etc.
3. **Engine Curves**
   - If `autoGenerateEngineRPMCurve` is true, editing `maxEngineTorque`, `maxEngineTorqueAtRPM`, or `min/maxEngineRPM` triggers dynamic rebuild of `engineTorqueCurve`.
4. **Transmission**
   - `automaticGear` is typical for simpler use. Semi-auto or manual modes require gear shifting calls from UI or inputs.
5. **Audio**
   - Use `AudioType.Off` to disable built-in engine sounds if you want to provide a custom system.
6. **Damage**
   - For mesh deformation, ensure your vehicle's mesh is marked as read/write in Unity's import settings. Otherwise, no deformation.
7. **Stability**
   - For tall vehicles, use stronger `antiRollVertical` or set `COM` lower.

    - If facing spin-out issues at high speed, `angularDragHelper` or `steeringHelper` help reduce uncontrollable spins.
8. **Optimization**
    - Many sub-features (damage, fuel, heat) can be disabled if not needed.

---

12. Summary

`RCC_CarControllerV4` is the **core** driving script in the Realistic Car Controller framework, unifying:

- **Wheel colliders**
- **Engine + torque**
- **Gears + transmissions**
- **Steering + drift**
- **Fuel + engine heat**
- **Audio**
- **Damage**

Through an array of **public fields** and **private methods**, it orchestrates realistic car physics, user inputs, driver assist systems, and optional advanced features like NOS, turbo, or collision-based mesh deformation. By customizing settings in this script (or letting it inherit from RCC's global behavior presets), one can easily create a wide variety of vehicle handling experiences.