Table of Contents

---

1. Overview

`RCC_UI_DashboardDisplay` manages the **HUD** or **dashboard** elements of a Realistic Car Controller (RCC) vehicle. It reads values from `RCC_DashboardInputs` (which reflect the current vehicle's status) and updates various UI panels, labels, and indicators in real time.

Typical usage includes speed and RPM readouts, gear display, ABS/ESP/handbrake lights, turn signal indicators, fuel, and engine heat warnings. It also manages separate display states (Full, Customization, TopButtonsOnly, Off) and can optionally integrate with Photon networking.

---

2. Class Declaration

```
[RequireComponent(typeof(RCC_DashboardInputs))]
public class RCC_UI_DashboardDisplay : RCC_Core {
    // ...
```

}

- Inherits from `RCC_Core`, granting access to `CarController`, `Settings`, etc.
- **`RequireComponent(typeof(RCC_DashboardInputs))`** ensures a `RCC_DashboardInputs` script is present on the same GameObject.

---

3. Purpose and Functionality

1. **Auto-finds** a vehicle and displays relevant dashboard info.
2. Provides **UI panels** for controller buttons, gauges, or customization menus.
3. Updates speed, RPM, gear text, as well as indicators for headlights, ABS, ESP, etc.
4. Lets you switch between different **display modes**—hiding or showing certain UI elements.

---

4. Fields and Properties

4.1 UI References and Panels

- **`public GameObject controllerButtons`**
  Container for on-screen or mobile control buttons.

- **`public GameObject gauges`**
  Container for speed, RPM, and other gauge displays.

- **`public GameObject customizationMenu`**
  UI panel for vehicle customization.

- **`public GameObject optionsMenu`**
  A general options panel, toggled in some scenarios.

- **`public GameObject spawn`**

- **`public GameObject spawn_Photon`**
  Buttons for spawning vehicles locally or over Photon networking.

- **`public TextMeshProUGUI RPMLabel, KMHLabel, GearLabel, recordingLabel`**
  Text fields for various readouts.

- **`public Image ABS, ESP, Park, Headlights, leftIndicator, rightIndicator, heatIndicator, fuelIndicator, rpmIndicator`**
  UI images for lighting up or changing color based on vehicle states.

- **`public Color color_On, color_Off`**
  Colors used for toggling on/off states (e.g. ABS, ESP).

- **`public Dropdown mobileControllersDropdown, carSelectionDropdown, carSelectionDropdown_Photon`**
  UI dropdowns for selecting mobile control type or vehicle in singleplayer/multiplayer modes.

4.2 Display Type Enumeration

public enum DisplayType { Full, Customization, TopButtonsOnly, Off }
public DisplayType displayType = DisplayType.Full;

- **`Full`**: Shows both controller buttons and gauges.
- **`Customization`**: Hides controller/gauges and shows the customization panel.
- **`TopButtonsOnly`**: Hides everything except possibly top-level or external buttons.
- **`Off`**: Disables all UI elements.

4.3 Vehicle Reference and Options

- **`public RCC_CarControllerV4 vehicle;`**
  The currently displayed vehicle.

- **`public bool findPlayerVehicleAuto = true;`**
  If enabled, automatically uses `RCCSceneManager.activePlayerVehicle`.

- **`public bool usePhotonWithThisCanvas = false;`** *(only if `PHOTON_UNITY_NETWORKING` and `RCC_PHOTON` are defined)*
  Switches certain UI elements for Photon-based multiplayer.

---

5. Initialization and Setup

- In **`Awake()`**, if Photon is enabled and `usePhotonWithThisCanvas` is true:
  - Switches from local to Photon-based UI elements (spawn buttons, car selection dropdown).
  - Potentially hides or shows certain UI for photon rooms.

The script references `RCC_DashboardInputs` on the same object via a private property:

```
 private RCC_DashboardInputs Inputs {
    get {
        if (inputs == null)
            inputs = GetComponent<RCC_DashboardInputs>();
        return inputs;
    }
}
```

-

---

6. Update Flow

6.1 Regular `Update()`

1. Checks if **`mobileControllersDropdown`** is interactable based on `Settings.mobileControllerEnabled`.
2. Toggles the correct panels (`controllerButtons`, `gauges`, `customizationMenu`) according to **`displayType`**:
    - **Full**: Show controller + gauges, hide customization.
    - **Customization**: Hide controller + gauges, show customization.
    - **TopButtonsOnly**: Hide everything except top buttons.
    - **Off**: Hide all.

6.2 `LateUpdate()`

1. Ensures **`RCC_DashboardInputs`** is enabled; if not, return early.
2. If `findPlayerVehicleAuto` is true, sets `vehicle = RCCSceneManager.activePlayerVehicle`.
3. If `vehicle` is `null`, returns early.
4. Updates **RPMLabel**, **KMHLabel**, **GearLabel**, etc. with values from `Inputs`.
5. Reflects the current RCC record/replay state (`RCC_SceneManager.recordMode`) in `recordingLabel`.
6. Changes colors of **ABS**, **ESP**, **Park**, **Headlights** images based on booleans in `Inputs`.
7. Manages **heatIndicator**, **fuelIndicator**, **rpmIndicator** colors for engine heat, fuel level, and near-redline RPM.
8. Displays **turn signals** (left, right, or hazard) by coloring `leftIndicator` and `rightIndicator` images.

---

7. Display Type Logic

The `displayType` affects which UI panels remain active:

Full:
controllerButtons.SetActive(true);
gauges.SetActive(true);
customizationMenu.SetActive(false);

-

Customization:
controllerButtons.SetActive(false);
gauges.SetActive(false);
customizationMenu.SetActive(true);

-

TopButtonsOnly:
controllerButtons.SetActive(false);
gauges.SetActive(false);
customizationMenu.SetActive(false);

-

Off:
controllerButtons.SetActive(false);
gauges.SetActive(false);
customizationMenu.SetActive(false);

-

---

8. Dashboard Element Updates

8.1 RPM, KMH, Gear, Recording State

- **`RPMLabel.text = Inputs.RPM.ToString("0");`**
- **`KMHLabel.text`** depends on `Settings.units` (KMH vs. MPH).
- **`GearLabel.text`** shows gear + 1 if forward, “R” if reverse, or “N” if neutral/changing.
- **`recordingLabel`** displays “Playing” (green) or “Recording” (red) if `RCCSceneManager.recordMode` is in Play/Record.

8.2 ABS, ESP, Park, Headlights

- Each is an Image that toggles color between `color_On` and `color_Off` based on `Inputs.ABS`, `Inputs.ESP`, etc.
- For `Park`, color is `Color.red` if engaged, otherwise `color_Off`.
- For `Headlights`, color is `Color.green` if on, otherwise `color_Off`.

8.3 Engine Heat, Fuel, RPM Indicators

- If `vehicle.engineHeat >= 100f`, **heatIndicator** is red; else it's a darker color.
- If `vehicle.fuelTank < 10f`, **fuelIndicator** is red; else it's a darker color.
- If `vehicle.engineRPM >= vehicle.maxEngineRPM - 500f`, **rpmIndicator** is red; else a darker color.

8.4 Turn Signals/Indicators

- For **Left**, **Right**, or **All** indicators, `leftIndicator` and `rightIndicator` set to an orange vs. dimmer color.
- For **Off**, both are dim.

---

9. Setting the Display Type

```
public void SetDisplayType(DisplayType _displayType) {
    displayType = _displayType;
}
```

- Allows external scripts (UI, triggers, etc.) to switch the canvas layout.

---

10. Usage Notes and Best Practices

1. **Attach** `RCC_UI_DashboardDisplay` to a Canvas or UI panel that also has an `RCC_DashboardInputs` script.
2. **Reference** the relevant UI elements in the Inspector (TextMeshPro fields, Images, Buttons, etc.).
3. **`findPlayerVehicleAuto`** is typically **true** for single-player setups; if you want a custom or AI vehicle, set it to **false** and assign `vehicle` manually.
4. **Photon** Support
    - If `usePhotonWithThisCanvas` is true, the script toggles certain UI for photon usage, including different spawn buttons.
    - Waits for `PhotonNetwork.InRoom` before enabling the options menu.
5. **Customization Menu**

- This script only toggles the panel's visibility; you'd handle actual upgrade logic with `RCC_Customization_API` or custom scripts.

---

11. Summary

`RCC_UI_DashboardDisplay` is a **dashboard controller** for Realistic Car Controller's UI, showing speed, RPM, gear, and warning lights on a **Canvas**. It integrates with `RCC_DashboardInputs` to pull current vehicle status and automatically updates relevant UI elements. The **DisplayType** enum provides flexibility in how much of the UI is visible at any moment (full gauge set, only top buttons, no UI, or a customization panel). It also supports **Photon** toggles for multiplayer scenarios.