
Table of Contents

---

1. Overview

`RCC_Customizer` is a high-level **vehicle customization** controller for **Realistic Car Controller (RCC)**. It serves as a container and manager for multiple sub-components (paint, wheels, spoilers, sirens, etc.), consolidating them into a single **loadout** system. This allows for:

- **Saving** and **loading** a vehicle's customization to/from PlayerPrefs (JSON).
- **Applying** or **restoring** upgrades (e.g., paint color, wheel selection).
- **Auto-initializing** each sub-manager.

It's typically attached to the same GameObject as an `RCC_CarControllerV4` or nested in a child, giving you a complete, modular system for in-game tuning and visuals.

---

2. Class Declaration

[DefaultExecutionOrder(10)]

```
public class RCC_Customizer : RCC_Core {

    // ...

}
```

- `[DefaultExecutionOrder(10)]` ensures it executes after some primary RCC scripts if needed.
- Inherits from `RCC_Core`, granting access to shared RCC methods and references.

---

3. Key Responsibilities

1. **Initialize** various customization managers (paint, wheels, spoilers, sirens, etc.).
2. **Handle loadouts** via a `RCC_Customizer_Loadout` object:
    - *AutoLoad* from disk at startup if desired.
    - *Save/Load/Delete* functionality.
3. **Auto-Save** changes if `autoSave` is enabled.
4. **Refresh** managers after changes (like picking a new paint color or spoiler).

---

4. Primary Fields and Properties

- `public string saveFileName`: Key for saving in `PlayerPrefs`. Usually set to the vehicle's name.
- `public bool autoLoadLoadout`: If true, the script loads existing customizations from PlayerPrefs upon initialization.
- `public bool autoSave`: If true, the script automatically saves modifications (not currently shown, but conceptually possible).
- `public RCC_Customizer_Loadout loadout`: A data struct representing all current customization settings (paint color, wheels, spoiler index, etc.).

4.1 Initialize Method

```
public enum InitializeMethod { Awake, OnEnable, Start, DelayedWithFixedUpdate, None }

public InitializeMethod initializeMethod = InitializeMethod.Start;
```

Determines **when** managers are initialized. Options:

- **Awake**: Right on `Awake()`.

- **OnEnable**: On `OnEnable()`.
- **Start**: In `Start()`.
- **DelayedWithFixedUpdate**: After a `FixedUpdate()` call.
- **None**: Don't auto-initialize; you can call `Initialize()` manually.

4.2 Loadout System

- **`public RCC_Customizer_Loadout loadout`**: A serializable class that can store the vehicle's chosen paint color, wheels, and upgrades.
- The script can **save** this loadout via JSON in `PlayerPrefs`, or **load** it from the same.

4.3 Customization Managers

The script references many possible sub-managers:

- **`PaintManager`** (`RCC_Customizer_PaintManager`)
- **`WheelManager`** (`RCC_Customizer_WheelManager`)
- **`UpgradeManager`** (`RCC_Customizer_UpgradeManager`)
- **`SpoilerManager`** (`RCC_Customizer_SpoilerManager`)
- **`SirenManager`** (`RCC_Customizer_SirenManager`)
- **`CustomizationManager`** (`RCC_Customizer_CustomizationManager`)
- **`DecalManager`** (`RCC_Customizer_DecalManager`)
- **`NeonManager`** (`RCC_Customizer_NeonManager`)

They are lazily loaded in properties, each searching in child transforms for the relevant component.

4.4 Saving and Loading

- **`Save()`**: Saves `loadout` to `PlayerPrefs` with `saveFileName` as the key, in JSON format.
- **`Load()`**: Attempts to read from `PlayerPrefs` and parse JSON back into `loadout`.
- **`Delete()`**: Clears that key from `PlayerPrefs` and also calls **`Restore()`** on all sub-managers, returning them to default.

---

5. Workflow / Lifecycle

5.1 Awake()

- If `initializeMethod = Awake`, calls `Initialize()`.
- If `autoLoadLoadout = true`, calls **`Load()`** first.

5.2 OnEnable()

- If `initializeMethod = OnEnable`, calls `Initialize()`.

5.3 Start()

- If `initializeMethod = Start`, calls `Initialize()`.
- If `initializeMethod = DelayedWithFixedUpdate`, uses a coroutine to wait for a `FixedUpdate()` then calls `Initialize()`.

---

6. Initializing the Managers

`Initialize()`:

1. Ensures **`loadout`** is non-null.
2. Calls `manager.Initialize()` on each manager if it exists (PaintManager, WheelManager, etc.).

Each manager typically reads or applies settings from `loadout`.

---

7. Saving, Loading, and Deleting Loadouts

- **`Save()`**: Serializes `loadout` to `JSON` and stores in `PlayerPrefs[saveFileName]`.
- **`Load()`**: If `PlayerPrefs.HasKey(saveFileName)`, reads it into `loadout`.
- **`Delete()`**: `PlayerPrefs.DeleteKey(saveFileName)` and calls `Restore()` on each manager, returning them to default.

---

8. Refreshing Upgrades

```
public void Refresh(MonoBehaviour component) {

    loadout.UpdateLoadout(component);

}
```

- A utility method: after you change an upgrade (like paint color or wheel type), you call `Refresh()` with the relevant manager.
- That manager updates the loadout with new data (like new color or spoiler index).

---

9. Hiding and Showing Visual Components

- `HideAll()`: Disables all visual items from SpoilerManager, SirenManager, DecalManager, NeonManager.
- `ShowAll()`: Re-enables them.
- Useful for toggling upgrade previews or a simpler camera view.

---

10. Usage Notes and Best Practices

1. **Attach** `RCC_Customizer` to the same GameObject (or a child) as your `RCC_CarControllerV4`.
2. **Set** `saveFileName` in the inspector, typically to the car's name.
3. **Initialization** can be done manually by setting `initializeMethod = None` and calling `Initialize()` yourself.
4. **Managers** must be child objects: e.g. a `RCC_Customizer_PaintManager` script on a "PaintManager" child.
5. **AutoLoad**
    - If `autoLoadLoadout = true` and there's a PlayerPrefs entry under `saveFileName`, it automatically applies it.
6. **AutoSave**
    - If you want to automatically save each time a user changes something, call `Save()` after a `Refresh()`.
7. **Deleting**
    - The `Delete()` method is a total reset, removing the save key and calling `Restore()` on each manager.

---

11. Summary

`RCC_Customizer` provides an **all-in-one** system for **vehicle customization** in Realistic Car Controller. By referencing multiple specialized **managers** (for paint, wheels, spoilers, etc.), it coordinates loading, saving, and applying upgrades. Through a **loadout** object, customizations are persistable between play sessions, offering players a consistent experience across game sessions. It simplifies modding each aspect of the vehicle's

appearance and performance, making it easy to integrate an in-game “customization garage” or menu.