Realistic Car Controller Pro (RCCP) – Complete Documentation

Contents

General Information

Realistic Car Controller Pro (RCCP) is a comprehensive vehicle physics system for Unity that provides realistic car simulation capabilities. The asset supports multiple render pipelines and platforms, making it versatile for various project requirements.

Supported Unity Versions

RCCP is compatible with Unity 2021.3 LTS and newer versions.

Compatible Render Pipelines

- Built-in Render Pipeline
- Universal Render Pipeline (URP)
- High Definition Render Pipeline (HDRP)

Supported Platforms

- PC (Windows, Mac, Linux)
- Mobile (Android, iOS)
- WebGL

Installation and Dependencies

Installing RCCP

1. Import the complete RCCP package from the Unity Asset Store
2. Select all assets in the importer window and click Import
3. The Welcome Window will appear automatically after import completion

Dependencies Installation

RCCP requires the New Input System package to function correctly. During installation:

1. The importer will prompt you to install dependencies
2. Choose "Yes" to install the New Input System automatically

3. This step is crucial for proper compilation

Required Layers

RCCP automatically adds these layers to your project:

- **RCCP_Vehicle** - Used on vehicle gameobjects
- **RCCP_WheelCollider** - Applied to wheel colliders
- **RCCP_DetachablePart** - For detachable vehicle parts
- **RCCP_Prop** - For destructible props (demo scenes)
- **RCCP_Crasher** - For damage scene machines (demo scenes)

Script Execution Order

The installation automatically configures the required script execution order to prevent event conflicts. This ensures proper component initialization and communication.

Scripting Symbols

The following symbols are automatically added to Player Settings:

- **BCG_RCCP** - Indicates RCCP installation
- **BCG_ENTEREXIT** - For Enter/Exit system integration
- **RCCP_PHOTON** - For Photon networking integration
- **BCG_URP** - When URP is detected

Opening the Prototype Demo Scene

If you choose not to import demo assets initially, you can still test RCCP functionality:

1. Open the RCCP Welcome Window
2. Click "Open Prototype Demo Scene"
3. This lightweight scene contains basic RCCP functionality without demo assets
4. Use this scene to verify installation and basic functionality

The prototype scene includes essential RCCP components and a basic vehicle setup for immediate testing.

RCCP Welcome Window

The Welcome Window is your central hub for RCCP configuration and scene management.

Accessing the Welcome Window

Navigate to **Tools → BoneCracker Games → Realistic Car Controller Pro → Welcome Window**

Welcome Window Features

- **Demo Scene Access** - Quick buttons to open various demo scenes
- **Addon Management** - Import additional features like Photon integration
- **Build Settings** - Automatically add demo scenes to build settings
- **Demo Content Management** - Import or delete demo assets
- **Quick Links** - Access to documentation and support resources

Important Functions

- **Add Demo Scenes to Build Settings** - Essential for running the AIO demo
- **Import Demo Assets** - Adds vehicles, scenes, and environment content
- **Delete Demo Content** - Removes all demo-related assets (irreversible)

Importing Demo Assets

Demo assets include vehicles, scenes, textures, and environment models that showcase RCCP capabilities.

Importing Process

1. Open the Welcome Window
2. Navigate to the "Addons" tab

3. Click "Import Demo Assets"
4. Confirm the import when prompted

What's Included

- **Vehicle Models** - Multiple car types with different configurations
- **Demo Scenes** - Various scenarios showcasing different features
- **Environment Assets** - City models, props, and destructible objects
- **Textures and Materials** - Visual assets for vehicles and environments

Build Size Consideration

Demo assets increase your build size significantly. If you don't plan to use them in your final project, consider deleting them after learning the system.

Opening Demo Scenes

Demo scenes showcase different RCCP features and serve as learning tools and testing environments.

Available Demo Scenes

- **RCCP City AIO** - All-in-one demonstration with multiple features
- **RCCP City** - Basic city driving environment
- **RCCP City Car Selection** - Vehicle selection and switching demo
- **RCCP Damage** - Vehicle damage system demonstration
- **RCCP Transport** - Trailer and transport mechanics
- **RCCP Blank** - Minimal setup for custom development
- **RCCP Customization** - Vehicle customization features
- **RCCP Override Inputs** - Custom input handling examples

Opening Scenes

1. Access scenes through the Welcome Window

2. Click the desired scene button
3. The scene will open automatically in the Scene Manager
4. Ensure scenes are added to Build Settings for runtime functionality

---

Scene Setup and System Overview

RCCP uses a modular component-based architecture where each vehicle system operates independently while communicating through events.

Core Architecture

- **Component-Based Design** - Each vehicle system (engine, transmission, etc.) is a separate component
- **Event-Driven Communication** - Components communicate through output events
- **Modular Configuration** - Add or remove components as needed
- **Performance Optimization** - LOD system manages performance based on distance

System Flow

1. **Input Reception** - RCCP_InputManager receives player inputs
2. **Input Distribution** - RCCP_Input component feeds inputs to vehicle systems
3. **Torque Generation** - Engine produces torque based on inputs
4. **Power Distribution** - Torque flows through transmission components
5. **Wheel Application** - Final torque is applied to wheel colliders
6. **Physics Simulation** - Unity physics handles vehicle movement
7. **Visual Updates** - Audio, particles, and visual effects respond to physics

---

RCCP Scene Manager

RCCP_SceneManager is automatically created in every scene and serves as the central management system for all RCCP-related components.

Primary Functions

- **Vehicle Management** - Tracks all vehicles in the scene
- **Player Vehicle Registration** - Manages which vehicle is player-controlled
- **Camera Coordination** - Links active camera to player vehicle
- **UI Synchronization** - Connects dashboard UI to active vehicle
- **Terrain Data Collection** - Gathers terrain information for ground materials

Key Properties

- activePlayerVehicle - Currently controlled vehicle
- activePlayerCamera - Camera following the player vehicle
- activePlayerCanvas - UI canvas displaying vehicle information
- allVehicles - List of all RCCP vehicles in the scene

Scene Manager Configuration

- **Register Last Vehicle as Player** - Automatically makes the most recently spawned vehicle controllable
- **Disable UI When No Player Vehicle** - Hides UI when no player vehicle exists
- **Terrain Processing** - Automatically processes terrain data for ground material system

Event Management

The Scene Manager listens for vehicle lifecycle events:

- Vehicle spawn/destroy events
- Camera creation events
- UI canvas initialization
- Player vehicle registration changes

Vehicle System - RCCP Car Controller

RCCP_CarController is the main component that unifies all vehicle systems and serves as the central hub for vehicle functionality.

Core Responsibilities

- **Component Coordination** - Manages all attached vehicle components
- **Physics Integration** - Interfaces with Unity's Rigidbody system
- **State Management** - Controls vehicle states (engine on/off, controllable, etc.)
- **Event Broadcasting** - Dispatches vehicle events to the scene manager

Essential Properties

- canControl - Determines if the vehicle accepts player input
- externalControl - Indicates external control (AI, networking, etc.)
- engineRunning - Current engine state
- rigid - Reference to the vehicle's Rigidbody component

Initialization Process

1. **Component Discovery** - Automatically finds all RCCP components
2. **Reference Assignment** - Links components to the car controller
3. **Event Registration** - Sets up component communication
4. **Physics Setup** - Configures rigidbody and colliders
5. **State Initialization** - Sets initial vehicle state

Vehicle Control Methods

- SetCanControl(bool) - Enable/disable player control
- StartEngine() - Start the engine
- KillEngine() - Stop the engine
- GetAllComponents() - Refresh component references

Addon Components Overview

RCCP uses a modular addon system where each vehicle function is handled by specialized components.

Core Drivetrain Components

RCCP_Engine

Generates torque based on throttle input and RPM. Features include:

- RPM simulation with realistic curves
- Torque output based on engine characteristics
- Engine sound generation
- Rev limiter functionality
- Turbo and supercharger simulation

RCCP_Clutch

Manages power transfer between engine and transmission:

- Clutch engagement simulation
- Slip calculation for realistic behavior
- Automatic and manual clutch modes
- Stall prevention mechanisms

RCCP_Gearbox

Handles gear ratios and transmission logic:

- Multiple gear ratios for different speeds
- Automatic and manual transmission modes
- Gear change timing and smoothness
- Reverse gear functionality

RCCP_Differential

Distributes power between wheels:

- Limited slip differential simulation
- Power distribution ratios
- Traction control integration
- Support for multiple differentials

RCCP_Axles

Manages wheel groups and axle behavior:

- Front and rear axle configuration
- Power distribution to wheels
- Brake force application
- Steering angle control

Vehicle Control Components

RCCP_Input

Receives input from RCCP_InputManager and distributes to vehicle systems:

- Input processing and filtering
- Mobile input integration
- Custom input override support
- AI input compatibility

RCCP_Stability

Provides electronic stability systems:

- Anti-lock braking system (ABS)
- Traction control system (TCS)
- Electronic stability program (ESP)

- Customizable intervention levels

RCCP_AeroDynamics

Simulates aerodynamic forces:

- Drag force calculation
- Downforce generation
- Air resistance at different speeds
- Realistic high-speed behavior

Visual and Audio Components

RCCP_Lights

Manages all vehicle lighting:

- Headlights with multiple modes
- Brake lights and turn signals
- Interior lighting
- Light intensity and color control

RCCP_Audio

Handles all vehicle audio:

- Engine sound simulation
- Transmission and differential sounds
- Brake and skid audio
- Environmental audio effects

RCCP_Particles

Controls particle effects:

- Wheel smoke and dust
- Engine exhaust particles

- Ground-specific particle effects
- Performance-optimized emission

Optional Addon Components

RCCP_Damage

Simulates vehicle damage:

- Collision damage calculation
- Deformable body parts
- Performance impact from damage
- Repair functionality

RCCP_Customizer

Enables vehicle customization:

- Paint color changes
- Wheel modifications
- Performance upgrades
- Visual modifications

RCCP_Nos

Nitrous oxide system:

- Temporary power boost
- Limited usage with refill mechanics
- Visual and audio effects
- Configurable boost parameters

Navigating the Car Controller Panel

The RCCP_CarController inspector provides comprehensive vehicle configuration through an organized panel system.

Panel Structure

The inspector is divided into sections for easy navigation:

- **General Settings** - Basic vehicle configuration
- **Drivetrain Components** - Engine, transmission, and power delivery
- **Control Systems** - Input, stability, and driver aids
- **Visual/Audio** - Lights, sounds, and particle effects
- **Optional Addons** - Additional features and customization

Adding Components

1. Locate the desired component section
2. Click the "Add" button next to the component name
3. Configure the newly added component
4. Check component integrity using the "Check" button

Removing Components

1. Select the component you want to remove
2. Use the "X" button or disable the component
3. Remove unused component references

Component Validation

Each component includes a "Check" button that:

- Validates component configuration
- Identifies missing references
- Reports potential issues
- Provides configuration recommendations

Best Practices

- Always use the "Check" buttons after making changes
- Ensure output events are properly connected
- Verify component dependencies before removal
- Test vehicle functionality after modifications

Enter Exit System

The Enter/Exit system allows players to transition between on-foot and vehicle control modes, enabling First Person Shooter (FPS) and Third Person Shooter (TPS) gameplay integration.

Installation

1. Open the RCCP Welcome Window
2. Navigate to the "Addons" tab
3. Import "BCG Shared Assets (Enter Exit System)"
4. The system will be integrated automatically

System Components

- **Character Controller** - Manages player movement when on foot
- **Vehicle Entry Points** - Defines where players can enter vehicles
- **Camera Transition** - Smooth camera switching between modes
- **Input Switching** - Automatic input mode changes

Functionality

- **Enter Vehicles** - Walk up to any vehicle and press the interaction key
- **Exit Vehicles** - Leave vehicles while maintaining character position
- **Multiple Entry Points** - Configure driver and passenger positions
- **Smooth Transitions** - Seamless camera and control transitions

Configuration

- **Entry Distance** - How close the player must be to enter
- **Entry Prompts** - UI indicators for interactive vehicles
- **Exit Positions** - Where the character appears when exiting
- **Animation Integration** - Support for entry/exit animations

Demo Scenes

- **City Enter/Exit FPS** - First-person city exploration with vehicles
- **City Enter/Exit TPS** - Third-person city environment
- **Blank Enter/Exit** - Minimal setup for custom development

---

Toolbar Menu Items

RCCP provides extensive toolbar integration for quick access to configuration tools and utilities.

Main Menu Path

Tools → BoneCracker Games → Realistic Car Controller Pro

Configuration Options

- **Edit Settings** - Opens the main RCCP_Settings asset
- **Configure Ground Materials** - Ground physics configuration
- **Configure Input** - Input system setup
- **Configure Behavior** - Vehicle behavior presets

Demo and Content Management

- **Welcome Window** - Main RCCP interface
- **Demo Vehicles and Prefabs** - Vehicle configuration assets
- **Demo Scenes and Levels** - Scene management tools

- **Demo Content Deletion** - Remove demo assets

Development Tools

- **Create Vehicle** - Vehicle creation wizard
- **Render Pipeline Converter** - Convert materials between render pipelines
- **Recordings** - Vehicle replay system configuration
- **Telemetry** - Performance monitoring tools

Integration Features

- **Photon Settings** - Networking configuration (if Photon is installed)
- **Enter/Exit Configuration** - Character integration settings

Quick Access Features

Most configuration assets can be accessed directly through:

- **GameObject Menu** - Right-click in hierarchy
- **Assets Menu** - Right-click in project window
- **Inspector Context** - Component-specific options

Asset Settings - RCCP Settings

RCCP_Settings is the central configuration asset that controls global vehicle behavior and system preferences.

Accessing Settings

Navigate to **Tools → BoneCracker Games → Realistic Car Controller Pro → Edit Settings**

General Configuration

- **Override Fixed TimeStep** - Custom physics timestep for consistent simulation
- **Override FPS** - Target framerate for optimal performance
- **Use Telemetry** - Enable real-time performance monitoring

- **Auto Save/Load Input Rebind** - Persist custom input configurations

Layer Configuration

Configure which layers RCCP uses for:

- Vehicle objects
- Wheel colliders
- Detachable parts
- Environmental props

Input System Selection

Choose between:

- **Old Input System** - Legacy Unity input (deprecated)
- **New Input System** - Modern Unity input system (recommended)

Mobile Configuration

- **Mobile Controller Type** - Choose mobile input interface style
- **Mobile UI Scale** - Adjust mobile interface sizing
- **Touch Sensitivity** - Configure touch input responsiveness

Behavior Settings

- **Default Behavior** - Starting vehicle behavior preset
- **Behavior Lock** - Prevent runtime behavior changes
- **Custom Behaviors** - Define custom vehicle presets

Camera Settings

- **Camera Switching** - Enable runtime camera mode changes
- **Default Camera Mode** - Starting camera perspective
- **Camera Smoothness** - Transition speed between camera modes

Performance Options

- **LOD System** - Enable distance-based performance optimization
- **Culling Distance** - Maximum distance for vehicle updates
- **Update Frequency** - How often to update non-critical systems

---

Mobile Controller Configuration

RCCP provides comprehensive mobile input support with customizable interface options.

Enabling Mobile Controls

1. Open RCCP_Settings
2. Navigate to Mobile Controller section
3. Select desired mobile controller type
4. Configure touch sensitivity and UI scaling

Mobile Controller Types

Simple Mobile

- Basic acceleration/brake buttons
- Simple steering wheel or tilt controls
- Minimal UI footprint
- Optimized for casual gameplay

Advanced Mobile

- Full vehicle control interface
- Gear selection buttons
- Handbrake and additional controls
- Comprehensive dashboard integration

Custom Mobile

- Fully customizable layout
- Drag-and-drop UI positioning
- Custom button configurations
- Project-specific interface design

Mobile Input Configuration

- **Steering Sensitivity** - How responsive steering feels
- **Acceleration Sensitivity** - Throttle response curve
- **Gyroscope Support** - Tilt-to-steer functionality
- **Haptic Feedback** - Vibration on supported devices

UI Customization

- **Button Sizes** - Scale mobile interface elements
- **Color Schemes** - Visual customization options
- **Transparency** - Interface opacity settings
- **Position Adjustment** - Move controls for different screen sizes

Platform-Specific Features

- **iOS Integration** - Game Center and iOS-specific features
- **Android Integration** - Google Play services integration
- **WebGL Optimization** - Browser-specific mobile controls

Behavior Types

RCCP includes preset vehicle behaviors that modify handling characteristics for different driving experiences.

Accessing Behavior Configuration

Open **Tools → BoneCracker Games → Realistic Car Controller Pro → Edit Settings → Behavior**

Default Behavior Types

Simulator

- Realistic physics simulation
- Accurate vehicle dynamics
- Challenging handling characteristics
- Suitable for racing simulators

Arcade

- Simplified physics for accessibility
- Enhanced grip and stability
- Reduced complexity for casual play
- Immediate responsiveness

Semi-Realistic

- Balanced between simulation and arcade
- Moderate realism with enhanced playability
- Good compromise for most projects
- Adjustable difficulty

Fun

- Exaggerated physics for entertainment
- Enhanced effects and feedback
- Dramatic handling characteristics
- Suitable for action games

Behavior Configuration

Each behavior type modifies:

- **Grip Levels** - Tire friction characteristics
- **Stability Settings** - Electronic aid intervention
- **Engine Response** - Power delivery characteristics
- **Steering Sensitivity** - Control responsiveness
- **Brake Force** - Stopping power and behavior

Custom Behaviors

Create custom behavior presets by:

1. Duplicating existing behavior settings
2. Modifying individual parameters
3. Saving as new behavior preset
4. Assigning to specific vehicles or scenes

Runtime Behavior Changes

Switch behaviors during gameplay using:

```
RCCP.SetBehavior(behaviorIndex);
```

Scene-Specific Behaviors

- Configure default behavior per scene
- Override individual vehicle behaviors
- Dynamic behavior switching based on game state

Ground Materials Configuration

RCCP's ground material system provides realistic surface interaction with different tire friction, particle effects, and audio for various terrain types.

Accessing Ground Materials

Navigate to **Tools → BoneCracker Games → Realistic Car Controller Pro → Configure Ground Materials**

System Overview

The ground material system works by detecting the PhysicMaterial of colliders that wheel colliders contact, then applying appropriate settings for that surface type.

Default Materials

RCCP includes three pre-configured materials:

1. **Asphalt** - High grip, minimal particles, road tire sounds
2. **Grass** - Medium grip, grass particles, softer audio
3. **Sand** - Low grip, sand particles, muffled tire sounds

Material Properties

Physics Settings

- **Forward Stiffness** - Longitudinal tire grip (acceleration/braking)
- **Sideways Stiffness** - Lateral tire grip (cornering)
- **Slip Threshold** - Minimum slip before particles activate
- **Damp** - Surface resistance that slows the vehicle

Audio Configuration

- **Ground Sound** - AudioClip for tire noise on this surface
- **Volume** - Maximum volume level for surface audio
- **Pitch Variation** - Sound variation based on wheel speed

Visual Effects

- **Ground Particles** - Particle system prefab for surface interaction
- **Skidmarks** - Custom skidmark texture for this material

- **Particle Emission Rate** - How many particles generate per slip amount

Creating New Materials

1. Click "Create New Ground Material"
2. Assign a PhysicMaterial to identify the surface
3. Configure physics properties for desired handling
4. Set up audio and particle effects
5. Test with vehicles on surfaces using that PhysicMaterial

Terrain Materials

For Unity Terrains, the system works with splatmap indices:

1. Configure terrain materials in the Terrain Materials section
2. Assign PhysicMaterial to the terrain collider
3. Map texture indices to ground material types
4. System automatically detects which texture the wheel is on

Common Setup Issues

- **Material Not Detected** - Ensure colliders have the correct PhysicMaterial assigned
- **No Particles** - Verify vehicle has RCCP_Particles component
- **Audio Problems** - Check AudioClip assignment and volume levels
- **Terrain Issues** - Verify splatmap index configuration matches terrain textures

---

Demo Scenes System

RCCP includes multiple demo scenes that showcase different aspects of the system and serve as learning tools for implementation.

Demo Scene Types

RCCP City AIO (All-In-One)

- Comprehensive demonstration of all RCCP features
- Multiple vehicles with different configurations
- Complete city environment with interactive elements
- Vehicle switching and customization examples
- Best starting point for learning RCCP

RCCP City

- Basic city driving environment
- Single vehicle focus
- Standard vehicle controls and camera
- Good for testing basic functionality

RCCP City Car Selection

- Vehicle selection and switching mechanics
- Multiple vehicle spawning
- UI integration examples
- Player vehicle registration demonstration

RCCP Damage

- Vehicle damage system showcase
- Destructible environment elements
- Collision damage mechanics
- Repair system demonstration

RCCP Transport

- Trailer attachment and detachment

- Heavy vehicle simulation
- Cargo transport mechanics
- Multi-vehicle coordination

RCCP Blank

- Minimal scene setup
- Clean environment for custom development
- Essential RCCP components only
- Good starting point for new projects

RCCP Customization

- Vehicle customization interface
- Paint, wheels, and modification systems
- Upgrade mechanics demonstration
- Visual customization tools

Scene Configuration

All demo scenes include:

- **RCCP_SceneManager** - Central scene management
- **RCCP_Camera** - Camera system with multiple modes
- **RCCP_Canvas** - UI system for vehicle information
- **Ground Setup** - Proper collision layers and materials

Using Demo Scenes

1. Open desired scene through Welcome Window
2. Enter Play Mode to test functionality
3. Use standard controls (WASD, mouse) for vehicle control
4. Experiment with different features and settings

5. Examine scene hierarchy to understand setup

Building Demo Scenes

To include demo scenes in builds:

1. Open Welcome Window
2. Click "Add Demo Scenes To Build Settings"
3. This automatically adds all demo scenes to the build list
4. Essential for runtime scene switching in AIO demo

Demo Vehicles Configuration

RCCP includes various pre-configured demo vehicles that showcase different vehicle types and configurations.

Accessing Demo Vehicles

Navigate to **Tools → BoneCracker Games → Realistic Car Controller Pro → Configuration → Demo Vehicles and Prefabs**

Vehicle Categories

Sports Cars

- High-performance configurations
- Advanced aerodynamics
- Sport-tuned suspension
- Racing-oriented setups

Sedans

- Balanced handling characteristics
- Comfort-oriented settings
- Daily driver configurations

- Moderate performance levels

Trucks and SUVs

- Heavy vehicle physics
- Higher center of gravity
- Increased towing capacity
- Off-road capabilities

Specialty Vehicles

- Unique configurations for specific purposes
- Experimental setups
- Demonstration of edge cases
- Advanced feature showcases

Vehicle Configuration Features

Each demo vehicle includes:

- **Complete Component Setup** - All necessary RCCP components
- **Optimized Settings** - Balanced performance and realism
- **Visual Integration** - Proper model and collider setup
- **Audio Configuration** - Appropriate engine and tire sounds

Customization Examples

Demo vehicles demonstrate:

- Different engine configurations
- Various transmission types
- Multiple suspension setups
- Different weight distributions
- Varied performance characteristics

Using Demo Vehicles as Templates

1. Duplicate desired demo vehicle prefab
2. Modify settings for your requirements
3. Replace 3D model with your own
4. Adjust component values as needed
5. Test and refine configuration

Performance Considerations

Demo vehicles are configured for:

- Balanced performance across platforms
- Reasonable physics complexity
- Good visual quality
- Stable handling characteristics

Render Pipeline Converter

RCCP includes a converter tool to help transition materials between different render pipelines.

Accessing the Converter

Navigate to **Tools → BoneCracker Games → Realistic Car Controller Pro → Render Pipeline Converter**

Conversion Types

Built-in to URP

- Converts Standard materials to URP Lit
- Adjusts texture assignments
- Maintains visual appearance
- Optimizes for URP performance

Built-in to HDRP

- Converts to HDRP Lit materials
- Enhanced visual quality options
- Advanced lighting features
- Performance considerations for HDRP

URP to HDRP

- Material upgrade path
- Enhanced visual features
- Quality improvements
- Feature compatibility checks

Conversion Process

1. Select materials to convert
2. Choose target render pipeline
3. Review conversion settings
4. Execute conversion
5. Test materials in target pipeline

Material Properties

The converter handles:

- **Albedo Textures** - Base color maps
- **Normal Maps** - Surface detail textures
- **Metallic/Roughness** - Material properties
- **Emission** - Self-illuminated materials
- **Transparency** - Alpha blending settings

Post-Conversion Steps

After conversion:

1. Verify material appearance
2. Adjust pipeline-specific settings
3. Test performance impact
4. Fine-tune visual quality
5. Update prefabs with new materials

Compatibility Notes

- **Decals and Neons** - Only compatible with URP and HDRP
- **Built-in Limitations** - Some effects not available in Built-in pipeline
- **Performance Impact** - Different pipelines have varying performance characteristics

---

Deleting Demo Content

RCCP provides a safe method to remove all demo content from your project to reduce build size and focus on your own content.

Deletion Process

1. Open the RCCP Welcome Window
2. Navigate to the bottom of the window
3. Click "Delete all demo contents from the project"
4. Confirm deletion in the dialog box

What Gets Deleted

The deletion removes:

- **Vehicle Models** - All demo vehicle 3D models
- **Vehicle Prefabs** - Pre-configured vehicle prefabs
- **Scene Assets** - Demo scene files

- **Environment Models** - City and prop models
- **Textures** - All demo-related textures
- **Audio Files** - Demo-specific audio clips
- **Materials** - Demo vehicle and environment materials

What Remains

Core RCCP functionality is preserved:

- **Core Scripts** - All RCCP component scripts
- **Settings Assets** - Configuration files
- **Example Prefabs** - Basic setup templates
- **Documentation** - User guides and references

Important Considerations

- **Irreversible Action** - Deletion cannot be undone
- **Build Size Reduction** - Significantly reduces project size
- **Lost References** - Any custom content referencing demo assets will break
- **Re-import Option** - Demo content can be re-imported from Welcome Window

Before Deletion Checklist

1. Backup your project
2. Verify you don't need demo content for reference
3. Confirm no custom scripts reference demo assets
4. Document any demo configurations you want to recreate

After Deletion

- Project will be lighter and faster to build
- Focus on your own content creation
- Use core RCCP components for custom vehicles

- Reference documentation for setup guidance

Base Classes

RCCP uses a hierarchical component system with base classes that provide common functionality and ensure proper integration.

RCCP_MainComponent

The core base class for all RCCP vehicle components.

Responsibilities

- **Component Registration** - Automatic discovery and registration with CarController
- **Reference Management** - Maintains references to other components
- **Lifecycle Management** - Handles component initialization and cleanup
- **Event Integration** - Provides event system connectivity

Key Properties

- CarController - Reference to the main RCCP_CarController
- Rigid - Vehicle's Rigidbody component
- Inputs - Input component reference
- Engine - Engine component reference
- AllWheelColliders - Array of all wheel colliders

Virtual Methods

- Awake() - Component initialization
- Start() - Component startup
- Update() - Per-frame updates
- FixedUpdate() - Physics updates

RCCP_Component

Specialized base class for standard vehicle components.

Features

- **Automatic Initialization** - Self-registers with CarController
- **Component Validation** - Built-in error checking
- **Performance Optimization** - Efficient update mechanisms
- **State Management** - Enable/disable functionality

Component Categories

- **Drivetrain Components** - Engine, transmission, differential
- **Control Components** - Input, stability, aerodynamics
- **Visual Components** - Lights, particles, audio
- **Optional Components** - Damage, customization, special features

RCCP_UpgradeComponent

Base class for components that support upgrade systems.

Upgrade Features

- **Performance Scaling** - Components can be upgraded for better performance
- **Level Management** - Multiple upgrade levels per component
- **Cost System** - Upgrade costs and requirements
- **Effect Application** - Automatic application of upgrade benefits

Supported Upgrades

- **Engine Upgrades** - Increased power and efficiency
- **Handling Upgrades** - Better grip and stability
- **Speed Upgrades** - Higher top speeds and acceleration

IRCCP_Component Interface

Interface that ensures proper component integration.

Required Implementation

- Initialize(RCCP_CarController) - Component setup with CarController reference
- GetComponentType() - Returns component type for registration
- OnValidate() - Component validation and error checking

Benefits

- **Type Safety** - Ensures proper component implementation
- **Standardization** - Consistent component behavior
- **Integration** - Seamless component communication

RCCP.cs Script and Static Methods

The RCCP.cs script provides a comprehensive API for runtime vehicle management, offering static methods for common operations.

Vehicle Spawning and Management

SpawnRCC Method

RCCP.SpawnRCC(vehiclePrefab, position, rotation, registerAsPlayerVehicle, isControllable, isEngineRunning)

Spawns a vehicle prefab with specified parameters:

- **vehiclePrefab** - RCCP_CarController prefab to instantiate
- **position** - World position for spawning
- **rotation** - Initial rotation
- **registerAsPlayerVehicle** - Whether to register as player vehicle
- **isControllable** - Enable player control
- **isEngineRunning** - Start with engine running

Vehicle Registration Methods

// Register vehicle as player vehicle

RCCP.RegisterPlayerVehicle(vehicle);

RCCP.RegisterPlayerVehicle(vehicle, isControllable);

RCCP.RegisterPlayerVehicle(vehicle, isControllable, engineState);

// Remove player vehicle registration

RCCP.DeRegisterPlayerVehicle();

Vehicle Control Methods

Control State Management

// Set controllable state

RCCP.SetControl(vehicle, isControllable);

// Set external control (AI, networking)

RCCP.SetExternalControl(vehicle, isExternal);

// Engine control

RCCP.SetEngine(vehicle, engineState);

Transmission Control

// Set automatic/manual transmission

RCCP.SetAutomaticGear(vehicle, isAutomatic);

Scene and Camera Management

Camera Operations

// Change camera mode

```
RCCP.ChangeCamera();
```

Vehicle Transportation

```
// Transport player vehicle
RCCP.Transport(position, rotation);

// Transport specific vehicle
RCCP.Transport(vehicle, position, rotation);
RCCP.Transport(vehicle, position, rotation, resetVelocity);
```

Behavior Management

```
// Change vehicle behavior
RCCP.SetBehavior(behaviorIndex);
```

Recording and Replay System

Recording Controls

```
// Start/stop recording
RCCP.StartStopRecord(vehicle);

// Start/stop replay
RCCP.StartStopReplay(vehicle);
RCCP.StartStopReplay(vehicle, recordedClip);

// Stop all recording/replay
RCCP.StopRecordReplay(vehicle);
```

Maintenance and Utilities

Vehicle Repair

```
// Repair specific vehicle
RCCP.Repair(vehicle);

// Repair player vehicle
RCCP.Repair();
```

Scene Cleanup

```
// Clean all skidmarks
RCCP.CleanSkidmarks();

// Clean specific skidmarks
RCCP.CleanSkidmarks(index);
```

Usage Examples

Basic Vehicle Spawning

```
// Spawn and register a player vehicle
RCCP_CarController newVehicle = RCCP.SpawnRCC(
    vehiclePrefab,
    spawnPosition,
    spawnRotation,
    true,  // Register as player
    true,  // Controllable
    true   // Engine running
);
```

Runtime Vehicle Switching

```
// Deregister current vehicle

RCCP.DeRegisterPlayerVehicle();


// Register new vehicle

RCCP.RegisterPlayerVehicle(newVehicle, true, true);
```

Input Override System

RCCP provides a flexible input override system that allows custom input handling while maintaining compatibility with the standard input system.

Understanding Input Flow

1. **RCCP_InputManager** - Receives raw input from devices
2. **RCCP_Input** - Vehicle component that processes inputs
3. **Vehicle Components** - Receive processed inputs for control

Input Override Methods

Direct Input Override

```
// Access vehicle input component

RCCP_Input vehicleInput = vehicle.GetComponent<RCCP_Input>();


// Override specific inputs

vehicleInput.throttleInput = customThrottleValue;

vehicleInput.steerInput = customSteerValue;

vehicleInput.brakeInput = customBrakeValue;
```

Custom Input Scripts

Create custom input handlers that:

1. Extend or replace RCCP_Input functionality
2. Process inputs from external sources
3. Maintain compatibility with vehicle systems
4. Support AI and networking integration

Input Properties

Available input overrides:

- **throttleInput** - Acceleration input (0 to 1)
- **brakeInput** - Brake input (0 to 1)
- **steerInput** - Steering input (-1 to 1)
- **handbrakeInput** - Handbrake input (0 to 1)
- **clutchInput** - Clutch input (0 to 1)
- **gearShiftUp** - Gear up command (bool)
- **gearShiftDown** - Gear down command (bool)

External Control Integration

AI Control

```
// Set vehicle to external control
RCCP.SetExternalControl(vehicle, true);


// AI script provides inputs
vehicleInput.throttleInput = aiThrottleDecision;
vehicleInput.steerInput = aiSteerDecision;
```

Network Integration

```
// Networked vehicle receives remote inputs
if (isRemotePlayer) {
   vehicleInput.throttleInput = networkThrottleInput;
   vehicleInput.steerInput = networkSteerInput;
}
```

Input Validation

The input system includes validation for:

- **Range Clamping** - Inputs are clamped to valid ranges
- **Smooth Transitions** - Prevents jarring input changes
- **Conflict Resolution** - Handles multiple input sources
- **Fallback Behavior** - Default values when inputs are invalid

Demo Implementation

The "RCCP Override Inputs" demo scene demonstrates:

- Custom input script implementation
- External input source integration
- Input validation and processing
- Seamless switching between input modes

Decals and Neons Compatibility

RCCP includes advanced visual features like decals and neon lighting effects that enhance vehicle appearance and customization options.

Render Pipeline Compatibility

- **URP (Universal Render Pipeline)** - Full support for decals and neons
- **HDRP (High Definition Render Pipeline)** - Full support with enhanced features

- **Built-in Render Pipeline** - Limited support, some features unavailable

Decal System

Features

- **Paint Decals** - Racing stripes, logos, and custom graphics
- **Damage Decals** - Scratches, dents, and wear marks
- **Dirt and Weathering** - Environmental effects on vehicle surfaces
- **Number Plates** - Customizable license plates and racing numbers

Implementation

Decals use:

- **Decal Projector** components for URP/HDRP
- **Mesh-based** solutions for Built-in pipeline
- **Runtime Application** - Add decals during gameplay
- **Texture Atlasing** - Efficient texture usage

Neon Lighting System

Features

- **Underglow Effects** - Customizable under-vehicle lighting
- **Interior Neons** - Dashboard and cabin accent lighting
- **Exterior Accents** - Hood, trunk, and body panel lighting
- **Color Customization** - Full RGB color control
- **Intensity Control** - Brightness and effect intensity

Technical Implementation

- **Emissive Materials** - Self-illuminated surfaces
- **Light Components** - Dynamic lighting effects

- **Particle Systems** - Enhanced glow effects
- **Shader Integration** - Custom shader support

Customization Integration

Neons and decals integrate with the RCCP_Customizer system:

- **Runtime Modification** - Change colors and effects during gameplay
- **Save/Load System** - Persist customizations between sessions
- **UI Integration** - User-friendly customization interfaces
- **Performance Optimization** - Efficient rendering for multiple effects

Performance Considerations

- **LOD Integration** - Effects disabled at distance
- **Quality Settings** - Scalable quality for different platforms
- **Batching Optimization** - Efficient rendering of multiple effects
- **Mobile Optimization** - Reduced complexity for mobile platforms

Setup Requirements

For optimal decal and neon functionality:

1. Use URP or HDRP render pipeline
2. Configure appropriate rendering layers
3. Set up proper lighting conditions
4. Optimize materials for target platform

Customizer System

The RCCP_Customizer component provides comprehensive vehicle customization capabilities, allowing runtime modification of vehicle appearance and performance.

Accessing the Customizer

The customizer is automatically included in demo vehicles and can be added to custom vehicles through the RCCP_CarController panel.

Customization Categories

Visual Customization

- **Paint Colors** - Body color modification with RGB or preset colors
- **Wheel Changes** - Different wheel designs and sizes
- **Window Tinting** - Adjustable window transparency and color
- **Decal Application** - Racing stripes, logos, and graphics
- **Neon Lighting** - Underglow and accent lighting effects

Performance Customization

- **Engine Upgrades** - Increased horsepower and torque
- **Handling Modifications** - Suspension and tire upgrades
- **Speed Enhancements** - Top speed and acceleration improvements
- **Brake Upgrades** - Enhanced stopping power
- **Weight Reduction** - Performance through weight optimization

Customization System Architecture

Runtime Modification

```
// Access customizer component
RCCP_Customizer customizer = vehicle.Customizer;


// Change paint color
customizer.ChangeColor(newColor);
```

```
// Apply wheel modification
customizer.ChangeWheels(wheelIndex);

// Enable/disable neon effects
customizer.SetNeonLighting(enabled, color);
```

Save and Load System

The customizer includes persistent storage:

- **Player Preferences** - Save customizations between sessions
- **JSON Export** - Export vehicle configurations
- **Profile System** - Multiple saved vehicle setups
- **Cloud Integration** - Optional cloud save functionality

UI Integration

The customizer works with UI systems:

- **Color Pickers** - Intuitive color selection interfaces
- **Preview System** - Real-time customization preview
- **Category Tabs** - Organized modification categories
- **Cost Display** - Show upgrade costs and requirements

Material and Asset Management

- **Material Swapping** - Runtime material changes
- **Mesh Replacement** - Different wheel and body part models
- **Texture Modification** - Dynamic texture application
- **Shader Parameters** - Real-time shader property changes

Performance Considerations

- **Asset Preloading** - Efficient customization asset management

- **Memory Management** - Optimal memory usage for customizations
- **Mobile Optimization** - Reduced complexity for mobile platforms
- **Quality Scaling** - Adjustable quality based on device capability

Demo Implementation

The customization demo scene showcases:

- Complete customization interface
- All modification categories
- Save/load functionality
- Performance upgrade effects
- Visual customization options

Upgrader System

The RCCP upgrader system allows performance modifications to vehicles through a structured upgrade progression system.

Upgrade Categories

Engine Upgrades

Progressive engine improvements:

- **Stage 1** - Basic power increase (10-20% improvement)
- **Stage 2** - Moderate enhancement (25-40% improvement)
- **Stage 3** - Significant upgrade (50-75% improvement)
- **Stage 4** - Maximum performance (100%+ improvement)

Each stage affects:

- **Maximum Power** - Peak horsepower and torque
- **Power Curve** - Torque delivery characteristics
- **Engine Response** - Throttle response speed

- **Engine Sound** - Audio modifications for upgraded engines

Handling Upgrades

Suspension and tire improvements:

- **Tire Compounds** - Different grip levels and wear characteristics
- **Suspension Tuning** - Adjustable stiffness and damping
- **Anti-Roll Bars** - Reduced body roll in corners
- **Aerodynamic Packages** - Increased downforce and stability

Speed Upgrades

Top speed and acceleration enhancements:

- **Transmission Optimization** - Better gear ratios
- **Aerodynamic Improvements** - Reduced drag coefficient
- **Weight Reduction** - Improved power-to-weight ratio
- **Turbo/Supercharger** - Forced induction systems

Upgrade Implementation

Component Integration

Upgrades integrate with RCCP components:

```
// Engine component receives upgrade
RCCP_Engine engine = vehicle.Engine;
engine.ApplyUpgrade(upgradeLevel);


// Handling components get improvements
RCCP_WheelCollider[] wheels = vehicle.AllWheelColliders;
foreach(var wheel in wheels) {
  wheel.ApplyHandlingUpgrade(upgradeLevel);
```

}

Progressive Scaling

Upgrades use mathematical scaling:

- **Linear Progression** - Consistent improvement per level
- **Exponential Curves** - Diminishing returns at higher levels
- **Custom Curves** - Tailored progression for different aspects
- **Balanced Scaling** - Maintains vehicle balance across upgrades

Economic System

The upgrade system includes cost management:

- **Currency System** - In-game currency for purchases
- **Unlock Requirements** - Prerequisites for higher-tier upgrades
- **Cost Scaling** - Increasing costs for better upgrades
- **Reward Integration** - Earn currency through gameplay

Visual Feedback

Upgrades provide visual indicators:

- **Performance Meters** - Display current upgrade levels
- **Visual Modifications** - Upgraded parts change appearance
- **Effect Particles** - Enhanced effects for upgraded vehicles
- **Audio Changes** - Different sounds for upgraded components

Upgrade Persistence

- **Save System** - Upgrades persist between sessions
- **Vehicle Profiles** - Different upgrade setups per vehicle
- **Backup/Restore** - Upgrade configuration management
- **Cloud Sync** - Optional cloud storage for upgrades

Demo Integration

The customization demo shows:

- Complete upgrade interface
- Progressive upgrade effects
- Cost and requirement systems
- Visual and audio feedback
- Performance impact demonstration

Spawning Vehicles at Runtime

RCCP provides comprehensive support for dynamic vehicle spawning during gameplay, essential for open-world games, vehicle selection systems, and multiplayer scenarios.

Basic Vehicle Spawning

Using RCCP.SpawnRCC Method

```
// Complete vehicle spawning with all options
RCCP_CarController spawnedVehicle = RCCP.SpawnRCC(
  vehiclePrefab,        // Vehicle prefab to spawn
  spawnPosition,        // World position
  spawnRotation,        // Initial rotation
  registerAsPlayerVehicle, // Register as player vehicle
  isControllable,       // Enable player control
  isEngineRunning       // Start engine state
);
```

Manual Spawning Process

```
// Traditional Unity instantiation
RCCP_CarController vehicle = Instantiate(vehiclePrefab, position, rotation);
```

```
// Manual registration and setup
vehicle.SetCanControl(true);
RCCP.RegisterPlayerVehicle(vehicle);
vehicle.StartEngine();
```

Spawn Point Management

Spawn Point System

Create spawn point arrays for organized vehicle placement:

```
[System.Serializable]
public class SpawnPoint {
    public Transform spawnTransform;
    public RCCP_CarController vehiclePrefab;
    public bool playerSpawn;
}

public SpawnPoint[] spawnPoints;
```

Random Spawning

```
// Random spawn point selection
SpawnPoint randomSpawn = spawnPoints[Random.Range(0, spawnPoints.Length)];
RCCP.SpawnRCC(
    randomSpawn.vehiclePrefab,
    randomSpawn.spawnTransform.position,
    randomSpawn.spawnTransform.rotation,
    randomSpawn.playerSpawn,
    true,
```

```
    true
);
```

Advanced Spawning Features

Conditional Spawning

```
// Spawn only if position is clear
if (IsSpawnPointClear(spawnPosition, vehicleSize)) {
    RCCP_CarController newVehicle = RCCP.SpawnRCC(
        vehiclePrefab, spawnPosition, spawnRotation,
        true, true, true
    );
}

bool IsSpawnPointClear(Vector3 position, Vector3 size) {
    Collider[] overlapping = Physics.OverlapBox(position, size * 0.5f);
    return overlapping.Length == 0;
}
```

Vehicle Pool System

For performance optimization:

```
public class VehiclePool : MonoBehaviour {
    public RCCP_CarController vehiclePrefab;
    public int poolSize = 10;
    private Queue<RCCP_CarController> vehiclePool;

    void Start() {
        InitializePool();
```

```
    }

    void InitializePool() {
        vehiclePool = new Queue<RCCP_CarController>();
        for (int i = 0; i < poolSize; i++) {
            RCCP_CarController vehicle = Instantiate(vehiclePrefab);
            vehicle.gameObject.SetActive(false);
            vehiclePool.Enqueue(vehicle);
        }
    }

    public RCCP_CarController SpawnFromPool(Vector3 position, Quaternion rotation) {
        if (vehiclePool.Count > 0) {
            RCCP_CarController vehicle = vehiclePool.Dequeue();
            vehicle.transform.position = position;
            vehicle.transform.rotation = rotation;
            vehicle.gameObject.SetActive(true);
            return vehicle;
        }
        return null;
    }
}
```

Vehicle Destruction and Cleanup

Proper Vehicle Destruction

```
// Deregister before destruction
```

```
if (vehicleToDestroy == RCCP_SceneManager.Instance.activePlayerVehicle) {
    RCCP.DeRegisterPlayerVehicle();
}

// Clean up vehicle
Destroy(vehicleToDestroy.gameObject);
```

Return to Pool

```
public void ReturnToPool(RCCP_CarController vehicle) {
    vehicle.SetCanControl(false);
    vehicle.gameObject.SetActive(false);
    vehiclePool.Enqueue(vehicle);
}
```

Multiplayer Spawning Considerations

- **Network Synchronization** - Ensure all players see spawned vehicles
- **Authority Management** - Control which player owns which vehicle
- **Spawn Coordination** - Prevent spawn point conflicts
- **State Synchronization** - Keep vehicle states synchronized

---

Vehicle Control State

RCCP provides comprehensive control over vehicle states, allowing dynamic enabling and disabling of player control for various gameplay scenarios.

Control State Management

Basic Control State

The canControl property determines whether a vehicle accepts player input:

```
// Enable player control
vehicle.SetCanControl(true);

// Disable player control
vehicle.SetCanControl(false);

// Check current control state
bool isControllable = vehicle.canControl;
```

Using RCCP Static Methods

```
// Set control state via API
RCCP.SetControl(vehicle, true);  // Enable control
RCCP.SetControl(vehicle, false); // Disable control
```

External Control System

External Control Flag

The externalControl property indicates when a vehicle is controlled by non-player systems:

```
// Set external control (AI, cutscenes, etc.)
RCCP.SetExternalControl(vehicle, true);

// Return to player control
RCCP.SetExternalControl(vehicle, false);
```

Control Priority System

Control states work hierarchically:

1. **External Control** - Highest priority (AI, cutscenes, networking)
2. **Can Control** - Player control permission

3. **Input Processing** - Actual input handling

Common Usage Scenarios

Cutscene Integration

```
// Start cutscene - disable player control
RCCP.SetControl(playerVehicle, false);
RCCP.SetExternalControl(playerVehicle, true);

// Cutscene AI takes control
cutsceneController.TakeControl(playerVehicle);

// End cutscene - return control to player
RCCP.SetExternalControl(playerVehicle, false);
RCCP.SetControl(playerVehicle, true);
```

Menu/Pause Systems

```
void PauseGame() {
  // Disable vehicle control during pause
  RCCP_CarController playerVehicle = RCCP_SceneManager.Instance.activePlayerVehicle;
  if (playerVehicle) {
    playerVehicle.SetCanControl(false);
  }
}

void ResumeGame() {
  // Re-enable vehicle control
  RCCP_CarController playerVehicle = RCCP_SceneManager.Instance.activePlayerVehicle;
```

```
    if (playerVehicle) {
        playerVehicle.SetCanControl(true);
    }
}
```

Vehicle Switching

```
void SwitchVehicle(RCCP_CarController newVehicle) {
    // Disable current vehicle
    RCCP_CarController currentVehicle = RCCP_SceneManager.Instance.activePlayerVehicle;
    if (currentVehicle) {
        currentVehicle.SetCanControl(false);
    }

    // Switch to new vehicle
    RCCP.RegisterPlayerVehicle(newVehicle, true);
}
```

AI Integration

AI Control Takeover

```
// AI component takes control
RCCP_AI aiComponent = vehicle.GetComponent<RCCP_AI>();
if (aiComponent) {
    RCCP.SetExternalControl(vehicle, true);
    aiComponent.enabled = true;
}
```

Hybrid Player/AI Control

```
// Temporary AI assistance (e.g., parking assist)
void EnableParkingAssist() {
    RCCP.SetExternalControl(playerVehicle, true);
    parkingAI.TakeTemporaryControl(playerVehicle);

    // Re-enable player control after completion
    StartCoroutine(ReturnControlAfterParking());
}
```

Network Multiplayer Considerations

```
// Local player vehicle
if (photonView.isMine) {
    vehicle.SetCanControl(true);
} else {
    // Remote player vehicle - no local control
    vehicle.SetCanControl(false);
    RCCP.SetExternalControl(vehicle, true);
}
```

State Validation

Always validate control states:

```
bool CanPlayerControlVehicle(RCCP_CarController vehicle) {
    return vehicle != null &&
           vehicle.canControl &&
           !vehicle.externalControl &&
           vehicle.gameObject.activeInHierarchy;
```

}

Input System Overview

RCCP features a comprehensive input system that supports both legacy and modern Unity input systems, with extensive customization options.

Input Architecture

RCCP_InputManager

The central input receiver that handles all player input:

- **Hidden GameObject** - Automatically created, not visible in hierarchy
- **Input Processing** - Receives raw input from devices
- **Input Distribution** - Sends processed input to vehicles
- **Platform Support** - Handles PC, mobile, and console inputs

RCCP_Input Component

Vehicle-level input processor:

- **Input Reception** - Receives input from RCCP_InputManager
- **Vehicle Integration** - Feeds input to vehicle components
- **Input Filtering** - Processes and smooths input values
- **Override Support** - Allows custom input sources

Supported Input Systems

New Input System (Recommended)

Modern Unity input system with advanced features:

- **Action-Based Input** - Flexible input action mapping
- **Device Support** - Comprehensive controller support
- **Rebinding** - Runtime input customization

- **Multiple Players** - Local multiplayer support

Legacy Input System

Traditional Unity input system:

- **Input Manager** - Classic Unity input configuration
- **Compatibility** - Works with older Unity versions
- **Simplicity** - Straightforward setup process

Input Selection

Configure input system in RCCP_Settings:

1. Open **Tools → BoneCracker Games → Realistic Car Controller Pro → Edit Settings**
2. Navigate to Input System section
3. Select desired input system
4. System automatically configures scripting symbols

Input Actions (New Input System)

Default Input Actions

RCCP includes pre-configured input actions:

- **Throttle** - Acceleration input
- **Brake** - Braking input
- **Steering** - Left/right steering
- **Handbrake** - Emergency brake
- **Gear Up/Down** - Manual transmission control
- **Camera Switch** - Change camera modes
- **Engine Start/Stop** - Engine control

Input Action Asset

RCCP uses RCCP_InputActions asset for configuration:

- **Action Maps** - Organized input groupings
- **Binding Paths** - Device-specific input mapping
- **Composite Bindings** - Complex input combinations
- **Control Schemes** - Different device configurations

Input Processing Flow

1. **Device Input** - Raw input from keyboard, gamepad, touch
2. **Input Actions** - Unity's input system processes actions
3. **RCCP_InputManager** - Receives processed input actions
4. **RCCP_Input** - Vehicle component processes vehicle-specific input
5. **Vehicle Components** - Components receive final input values

Mobile Input Integration

- **Touch Controls** - Screen-based vehicle controls
- **Gyroscope** - Tilt-to-steer functionality
- **Haptic Feedback** - Vibration on supported devices
- **Custom UI** - Customizable mobile interface

Input Validation and Smoothing

- **Range Clamping** - Ensures input values stay within valid ranges
- **Deadzone Handling** - Eliminates controller drift
- **Input Smoothing** - Reduces jarring input changes
- **Acceleration Curves** - Non-linear input response

Input Actions Editing

RCCP provides comprehensive input customization through Unity's Input Actions system, allowing players and developers to modify controls according to their preferences.

Accessing Input Actions

Input Actions Asset Location

The RCCP input actions are stored in: Assets/Realistic Car Controller Pro/Inputs/RCCP_InputActions.inputactions

Opening the Input Actions Editor

1. Select the RCCP_InputActions asset in the Project window
2. Click "Edit asset" in the Inspector
3. Unity's Input Actions editor opens

Input Actions Structure

Action Maps

RCCP organizes inputs into logical groups:

- **Vehicle** - Primary vehicle control actions
- **Camera** - Camera control and switching
- **UI** - User interface navigation

Vehicle Actions

Primary vehicle control inputs:

- **Throttle** - Forward acceleration
- **Brake** - Braking force
- **Steering** - Left/right steering control
- **Handbrake** - Emergency/parking brake
- **Clutch** - Manual transmission clutch
- **Gear Up** - Shift to higher gear
- **Gear Down** - Shift to lower gear
- **Engine** - Start/stop engine

Camera Actions

Camera control inputs:

- **Change Camera** - Cycle through camera modes
- **Look Around** - Free-look camera control
- **Zoom** - Camera zoom in/out

Customizing Input Bindings

Adding New Bindings

1. Select an action in the Input Actions editor
2. Click the "+" button next to "Bindings"
3. Choose binding type (Button, Axis, etc.)
4. Configure the new binding path

Modifying Existing Bindings

1. Select the binding to modify
2. Click the binding path field
3. Choose new input source from the dropdown
4. Configure additional binding properties

Composite Bindings

For complex inputs like WASD steering:

1. Add a new binding
2. Select "Add Up/Down/Left/Right Composite"
3. Configure each direction separately
4. Set appropriate binding paths

Control Schemes

Default Control Schemes

RCCP includes several control schemes:

- **Keyboard & Mouse** - Standard PC controls
- **Gamepad** - Xbox/PlayStation controller support
- **Touch** - Mobile touch controls

Creating Custom Control Schemes

1. Click "Control Schemes" in the Input Actions editor
2. Add new control scheme
3. Define required devices
4. Configure device requirements

Runtime Input Rebinding

Rebinding API

RCCP supports runtime input rebinding:

```
// Start rebinding process
InputAction action = inputActions.Vehicle.Steering;
InputActionRebindingExtensions.RebindingOperation rebindOperation =
    action.PerformInteractiveRebinding();

// Configure rebinding
rebindOperation
    .WithControlsExcluding("Mouse")
    .OnMatchWaitForAnother(0.1f)
    .OnComplete(operation => {
```

```
    // Rebinding completed
    operation.Dispose();
  })
  .Start();
```

Save/Load Rebindings

```
// Save rebindings
string rebinds = inputActions.SaveBindingOverridesAsJson();
PlayerPrefs.SetString("InputRebinds", rebinds);

// Load rebindings
string rebinds = PlayerPrefs.GetString("InputRebinds");
inputActions.LoadBindingOverridesFromJson(rebinds);
```

Input Action Properties

Action Types

- **Button** - Binary on/off input
- **Value** - Continuous value input
- **PassThrough** - Direct input passthrough

Binding Properties

- **Path** - Input device and control specification
- **Interactions** - How input triggers (Press, Hold, etc.)
- **Processors** - Input value modification (Scale, Invert, etc.)

Testing Input Changes

1. Apply changes in the Input Actions editor
2. Save the asset

3. Enter Play Mode to test new bindings
4. Use the Input Debugger to monitor input values

Platform-Specific Considerations

- **PC** - Keyboard, mouse, and gamepad support
- **Mobile** - Touch screen and accelerometer inputs
- **Console** - Platform-specific controller requirements
- **Web** - Browser input limitations and compatibility

UI Canvas System

RCCP includes a comprehensive UI system that provides real-time vehicle information and player interaction interfaces.

RCCP_UIManager Overview

Core Functionality

RCCP_UIManager serves as the central hub for all vehicle-related user interface elements:

- **Dashboard Integration** - Real-time vehicle data display
- **Mobile Controls** - Touch-based vehicle control interface
- **Menu Systems** - Settings and configuration interfaces
- **HUD Elements** - Speed, RPM, gear indicators

Automatic Integration

The UI system automatically integrates with:

- **Active Player Vehicle** - Displays data from current player vehicle
- **Scene Manager** - Responds to vehicle registration changes
- **Input System** - Provides mobile input alternatives
- **Settings System** - Reflects current RCCP configuration

Dashboard Components

Speedometer

Real-time speed display with multiple units:

- **KM/H** - Kilometers per hour
- **MPH** - Miles per hour
- **Digital Display** - Numerical speed readout
- **Analog Gauge** - Traditional needle-style speedometer

Tachometer (RPM Gauge)

Engine revolution monitoring:

- **RPM Range** - Configurable maximum RPM
- **Red Line** - Engine danger zone indication
- **Shift Lights** - Gear change recommendations
- **Engine State** - Visual engine on/off indication

Gear Indicator

Transmission state display:

- **Current Gear** - Active gear number or letter (R, N, P)
- **Transmission Mode** - Manual or automatic indication
- **Gear Recommendations** - Optimal shift point indicators
- **Clutch State** - Manual transmission clutch engagement

Additional Dashboard Elements

- **Fuel Gauge** - Fuel level indication (if fuel system enabled)
- **Temperature Gauge** - Engine temperature monitoring
- **Damage Indicator** - Vehicle condition status
- **Warning Lights** - Various vehicle system alerts

UI Canvas Configuration

Canvas Setup

RCCP_UIManager requires proper canvas configuration:

- **Render Mode** - Screen Space - Overlay (recommended)
- **Canvas Scaler** - UI Scale Mode set to "Scale With Screen Size"
- **Reference Resolution** - 1920x1080 recommended
- **Match Mode** - Match Width or Height as appropriate

Event System Integration

The UI system works with both input systems:

- **Legacy Input Module** - For old input system compatibility
- **Input System UI Input Module** - For new input system
- **Automatic Switching** - Based on RCCP_Settings configuration

Mobile UI Integration

Mobile Controller Display

When mobile controls are enabled:

- **Steering Wheel** - Touch-based steering control
- **Throttle/Brake** - Acceleration and brake pedals
- **Additional Controls** - Handbrake, gear shifting, camera controls
- **Customizable Layout** - Adjustable control positions and sizes

Responsive Design

The UI system adapts to different screen sizes:

- **Aspect Ratio Handling** - Maintains proportions across devices
- **Safe Area Support** - Respects device safe areas (notches, etc.)
- **Orientation Support** - Landscape and portrait modes

- **DPI Scaling** - Appropriate sizing for different screen densities

TextMeshPro Integration

Text Rendering

RCCP uses TextMeshPro for all text elements:

- **High Quality** - Sharp text rendering at all resolutions
- **Rich Text Support** - Formatting, colors, and effects
- **Localization Ready** - Easy text replacement for multiple languages
- **Performance Optimized** - Efficient text rendering

Font Configuration

- **Default Font** - RCCP includes optimized fonts
- **Custom Fonts** - Support for project-specific typography
- **Fallback Fonts** - Automatic fallback for missing characters
- **Font Asset Management** - Organized font asset structure

UI Customization

Theme System

Customize UI appearance:

- **Color Schemes** - Modify UI color palettes
- **Layout Adjustments** - Reposition UI elements
- **Scale Modifications** - Resize interface elements
- **Style Variations** - Different visual styles for UI components

Custom UI Elements

Add project-specific UI:

- **Additional Gauges** - Custom vehicle data displays

- **Game-Specific HUD** - Integration with gameplay systems
- **Menu Extensions** - Additional settings and options
- **Notification Systems** - In-game message displays

Performance Optimization

UI Performance

The UI system includes optimizations:

- **Update Frequency** - Configurable refresh rates for different elements
- **Canvas Groups** - Efficient enable/disable of UI sections
- **Object Pooling** - Reuse of temporary UI elements
- **Culling** - Hide off-screen UI elements

Mobile Optimization

Special considerations for mobile platforms:

- **Reduced Update Frequency** - Lower refresh rates for better performance
- **Simplified Effects** - Reduced visual complexity
- **Touch Optimization** - Larger touch targets and better responsiveness
- **Battery Conservation** - Efficient rendering to preserve battery life

---

Mobile Input Controls

RCCP provides comprehensive mobile input support through the RCCP_MobileInputs component, offering intuitive touch-based vehicle control.

RCCP_MobileInputs Component

Core Functionality

The mobile input system provides touch-based alternatives to keyboard/gamepad controls:

- **Touch Steering** - Steering wheel or tilt-based steering

- **Virtual Pedals** - Touch-based throttle and brake controls
- **Gesture Support** - Swipe and tap gestures for various functions
- **Haptic Feedback** - Vibration feedback on supported devices

Integration with Input System

Mobile inputs seamlessly integrate with RCCP's input architecture:

- **Input Manager Integration** - Feeds into RCCP_InputManager
- **Parallel Processing** - Works alongside other input methods
- **Priority System** - Mobile inputs take precedence when active
- **Fallback Support** - Automatic switching between input methods

Mobile Control Types

Steering Controls

Virtual Steering Wheel

- **Visual Steering Wheel** - On-screen steering wheel interface
- **Touch and Drag** - Rotate by touching and dragging
- **Return to Center** - Automatic centering when released
- **Customizable Sensitivity** - Adjustable steering response

Tilt Steering (Gyroscope)

- **Accelerometer Input** - Device tilt controls steering
- **Calibration System** - Auto-calibration for different holding positions
- **Sensitivity Settings** - Adjustable tilt responsiveness
- **Deadzone Configuration** - Eliminate unwanted micro-movements

Touch Steering

- **Left/Right Touch Areas** - Screen zones for steering input

- **Pressure Sensitivity** - Steering amount based on touch pressure
- **Multi-touch Support** - Multiple finger steering control

Acceleration and Braking

Virtual Pedals

- **Throttle Pedal** - Touch and hold for acceleration
- **Brake Pedal** - Separate brake control
- **Pressure Sensitivity** - Input strength based on touch pressure
- **Visual Feedback** - Pedal depression animation

Single Touch Controls

- **Combined Control** - Single touch area for gas/brake
- **Touch and Drag** - Vertical drag for throttle/brake control
- **Split Screen** - Left/right areas for different functions

Additional Controls

- **Handbrake Button** - Emergency brake activation
- **Gear Shift Controls** - Manual transmission gear selection
- **Camera Controls** - Touch-based camera switching
- **Horn/Lights** - Additional vehicle function controls

Mobile UI Configuration

Control Layout

Customize mobile interface layout:

- **Drag and Drop** - Reposition controls in editor
- **Scale Adjustment** - Resize controls for different screen sizes
- **Transparency Settings** - Adjust control visibility

- **Safe Area Handling** - Automatic adjustment for device notches

Visual Customization

- **Control Skins** - Different visual styles for controls
- **Color Themes** - Customize control colors
- **Animation Effects** - Control press and release animations
- **Particle Effects** - Visual feedback for control activation

Platform-Specific Features

iOS Integration

- **Game Center** - Achievement and leaderboard integration
- **Metal Rendering** - Optimized rendering for iOS devices
- **Touch ID/Face ID** - Biometric authentication support
- **Haptic Engine** - Advanced haptic feedback

Android Integration

- **Google Play Services** - Achievement and cloud save integration
- **Vulkan Rendering** - Modern graphics API support
- **Adaptive Icons** - Support for adaptive app icons
- **Android Haptics** - Platform-specific vibration patterns

Performance Optimization

Touch Input Optimization

- **Input Pooling** - Efficient touch event handling
- **Update Frequency** - Configurable touch update rates
- **Deadzone Processing** - Eliminate unnecessary input processing
- **Battery Conservation** - Optimized for mobile battery life

UI Performance

- **Canvas Optimization** - Efficient UI rendering
- **Texture Atlasing** - Optimized texture usage
- **Draw Call Reduction** - Minimized rendering overhead
- **Memory Management** - Efficient mobile memory usage

Mobile Settings Configuration

Sensitivity Settings

Configure mobile input responsiveness:

- **Steering Sensitivity** - How quickly steering responds
- **Acceleration Sensitivity** - Throttle response curve
- **Brake Sensitivity** - Braking force response
- **Gyroscope Sensitivity** - Tilt steering responsiveness

Comfort Settings

- **Haptic Feedback** - Enable/disable vibration
- **Auto-Center Steering** - Automatic steering return
- **Input Smoothing** - Reduce input jitter
- **Visual Feedback** - Control press indicators

Demo Implementation

The mobile demo scenes showcase:

- **Complete Mobile Interface** - Full touch control implementation
- **Different Control Schemes** - Various mobile input methods
- **Performance Optimization** - Mobile-specific optimizations
- **Platform Integration** - iOS and Android specific features

---

Trailer System

RCCP includes a comprehensive trailer system that enables realistic towing simulation with proper physics and connection mechanics.

RCCP_TrailerController Overview

Core Functionality

The trailer controller manages trailer-specific behavior:

- **Independent Physics** - Separate rigidbody simulation
- **Wheel Management** - Trailer wheel physics and alignment
- **Connection Management** - Attachment point handling
- **Stability Control** - Trailer-specific stability systems

Component Structure

RCCP_TrailerController includes:

- **Rigidbody Integration** - Physics simulation for trailer mass
- **Wheel Colliders** - Trailer wheel physics
- **Joint Management** - Connection joint configuration
- **Brake System** - Trailer brake integration

Trailer Physics

Mass and Inertia

Proper trailer physics simulation:

- **Realistic Mass** - Appropriate trailer weight distribution
- **Inertia Calculation** - Accurate rotational inertia
- **Center of Mass** - Proper weight balance point
- **Load Distribution** - Dynamic weight distribution based on cargo

Suspension System

Trailer-specific suspension:

- **Spring Configuration** - Appropriate for trailer loads
- **Damping Settings** - Stability-focused damping
- **Load Compensation** - Suspension adjustment for cargo weight
- **Articulation Points** - Proper trailer pivoting

Connection Mechanics

Attachment Points

Trailer connection system:

- **Hitch Position** - Proper connection point location
- **Joint Configuration** - Unity joint setup for realistic connection
- **Breakaway Simulation** - Connection failure under stress
- **Electrical Connection** - Trailer light integration

Joint Physics

Realistic trailer joint behavior:

- **Rotational Freedom** - Appropriate joint movement limits
- **Connection Strength** - Joint force limits
- **Oscillation Damping** - Reduced trailer swaying
- **Load Transfer** - Force transfer between vehicle and trailer

Trailer Types

Cargo Trailers

Standard cargo hauling:

- **Enclosed Trailers** - Protected cargo transport
- **Flatbed Trailers** - Open cargo platform

- **Container Trailers** - Standardized container transport
- **Specialized Trailers** - Custom cargo configurations

Recreational Trailers

Non-commercial trailers:

- **Travel Trailers** - RV and camping trailers
- **Boat Trailers** - Watercraft transport
- **Utility Trailers** - General purpose hauling
- **Equipment Trailers** - Specialized equipment transport

Trailer Configuration

Setup Process

1. **Add RCCP_TrailerController** - Attach component to trailer GameObject
2. **Configure Rigidbody** - Set appropriate mass and drag
3. **Setup Wheel Colliders** - Configure trailer wheels
4. **Connection Point** - Define hitch attachment location
5. **Physics Tuning** - Adjust suspension and stability

Component Requirements

Essential trailer components:

- **RCCP_TrailerController** - Main trailer management
- **Rigidbody** - Physics simulation
- **Wheel Colliders** - Trailer wheel physics
- **Colliders** - Collision detection
- **Lights (Optional)** - Trailer lighting system

Advanced Features

Multi-Trailer Support

Connect multiple trailers:

- **Trailer Chains** - Multiple trailer connections
- **Length Limits** - Realistic trailer train lengths
- **Stability Management** - Enhanced stability for longer combinations
- **Articulation Control** - Proper joint management

Dynamic Loading

Runtime cargo management:

- **Cargo Addition** - Add cargo during gameplay
- **Weight Distribution** - Dynamic weight calculation
- **Physics Updates** - Real-time physics adjustment
- **Visual Updates** - Cargo visualization

Performance Considerations

Physics Optimization

- **LOD System** - Reduced physics detail at distance
- **Update Frequency** - Configurable physics update rates
- **Culling Distance** - Maximum simulation distance
- **Component Pooling** - Efficient trailer instance management

Stability Enhancements

- **Damping Systems** - Reduced oscillation and swaying
- **Anti-Jackknife** - Prevention of dangerous trailer angles
- **Speed Limits** - Automatic speed reduction with trailers
- **Electronic Stability** - Trailer-specific stability control

Trailer Connector System

The RCCP_TrailerAttacher component provides the connection mechanism that allows vehicles to attach and detach trailers dynamically.

RCCP_TrailerAttacher Overview

Core Functionality

The trailer attacher manages the connection between vehicles and trailers:

- **Detection System** - Automatic trailer detection within range
- **Connection Management** - Attach and detach operations
- **Joint Creation** - Dynamic Unity joint management
- **Status Monitoring** - Connection state tracking

Component Integration

RCCP_TrailerAttacher integrates with:

- **Vehicle Physics** - Affects vehicle handling when trailer attached
- **Input System** - Provides attach/detach input controls
- **UI System** - Connection status indicators
- **Audio System** - Connection sound effects

Connection Detection

Range-Based Detection

Automatic trailer detection system:

- **Detection Radius** - Configurable detection distance
- **Angle Restrictions** - Valid connection angles
- **Height Tolerance** - Vertical alignment requirements
- **Obstacle Checking** - Clear path verification

Visual Indicators

Connection assistance features:

- **Proximity Indicators** - Visual cues for trailer distance
- **Alignment Guides** - Proper positioning assistance
- **Connection Status** - Visual connection state feedback
- **Range Visualization** - Editor gizmos for detection range

Connection Process

Automatic Connection

Simplified connection process:

```
// Automatic connection when in range
void Update() {
    if (IsTrailerInRange() && Input.GetKeyDown(attachKey)) {
        AttachNearestTrailer();
    }
}

void AttachNearestTrailer() {
    RCCP_TrailerController nearestTrailer = FindNearestTrailer();
    if (nearestTrailer != null) {
        CreateConnection(nearestTrailer);
    }
}
```

Manual Connection

Precise connection control:

```
// Manual connection with specific trailer
```

```
public void AttachTrailer(RCCP_TrailerController trailer) {
  if (CanAttachTrailer(trailer)) {
    CreateConnection(trailer);
    UpdateVehiclePhysics();
  }
}
```

Joint Configuration

Joint Types

Different connection joint options:

- **ConfigurableJoint** - Full 6-DOF control
- **HingeJoint** - Simple rotation-only connection
- **FixedJoint** - Rigid connection (unrealistic but stable)
- **Custom Joint** - Project-specific joint implementations

Joint Parameters

Configurable joint properties:

- **Connection Strength** - Joint break force limits
- **Rotation Limits** - Maximum connection angles
- **Damping Settings** - Oscillation reduction
- **Spring Settings** - Connection flexibility

Connection Management

Attachment Process

Step-by-step connection:

1. **Proximity Check** - Verify trailer is within range
2. **Alignment Verification** - Check proper positioning

3. **Joint Creation** - Create Unity joint connection
4. **Physics Update** - Adjust vehicle handling
5. **Status Update** - Update connection state
6. **Audio/Visual Feedback** - Connection confirmation

Detachment Process

Safe disconnection:

1. **Input Detection** - Detach command received
2. **Safety Check** - Verify safe detachment conditions
3. **Joint Destruction** - Remove Unity joint
4. **Physics Restoration** - Restore vehicle handling
5. **Status Update** - Update connection state
6. **Audio/Visual Feedback** - Disconnection confirmation

Vehicle Handling Modifications

Physics Adjustments

Changes when trailer attached:

- **Increased Mass** - Virtual mass increase for realistic handling
- **Reduced Acceleration** - Slower acceleration with trailer load
- **Extended Braking** - Longer stopping distances
- **Steering Modifications** - Reduced steering responsiveness

Performance Impact

Trailer effects on vehicle:

```
void UpdateVehiclePhysics() {
  if (hasTrailerAttached) {
    // Adjust vehicle performance
```

```
        engine.maxTorque *= trailerLoadFactor;
        stability.interventionThreshold *= trailerStabilityFactor;
        aerodynamics.dragCoefficient += trailerDragIncrease;
    }
}
```

Advanced Features

Multiple Hitch Points

Support for various connection types:

- **Rear Hitch** - Standard trailer connection
- **Front Mount** - Special purpose connections
- **Multi-Point** - Heavy-duty trailer connections
- **Quick-Release** - Fast connect/disconnect systems

Electrical Integration

Trailer electrical systems:

- **Light Synchronization** - Trailer lights follow vehicle
- **Brake Light Integration** - Trailer brake lights
- **Turn Signal Sync** - Trailer turn indicators
- **Electrical Failure** - Simulation of electrical problems

Safety Systems

Connection Safety

Safety features for trailer connections:

- **Speed Limitations** - Automatic speed reduction
- **Stability Monitoring** - Connection stress monitoring
- **Breakaway Detection** - Automatic disconnection under stress

- **Warning Systems** - Alerts for unsafe conditions

Emergency Procedures

Emergency disconnection:

```
void EmergencyDetach() {
  if (hasTrailerAttached && IsEmergencyCondition()) {
    DestroyJoint();
    PlayEmergencySound();
    ShowWarningMessage();
  }
}
```