Kerbal Space Program Wheel System – Complete Reference

This document provides a comprehensive reference for the KSP 1.x wheel system. Each wheel-related PartModule is explained field-by-field, followed by examples and a comparison between rover wheels and landing gear.

Note: This was written with the help of ChatGPT

ModuleWheelBase

Core module required for all wheels. It registers the part as a wheel, manages ground contact, damage state, and provides shared data for all other wheel modules.

Fields

- **wheelColliderTransformName:** Name of the WheelCollider transform used for physics calculations.
- **wheelTransformName:** Transform used for the visible wheel mesh.
- **wheelType:** Classification of the wheel (FREE, MOTORIZED, STEERABLE).
- **radius:** Physical wheel radius in meters.
- **center:** Local offset of the wheel collider center.
- **mass:** Effective mass of the wheel for physics calculations.
- **frictionMultiplier:** Global multiplier applied to wheel friction curves.

Example

```
MODULE
{
  name = ModuleWheelBase
  wheelColliderTransformName = WheelCollider
  wheelTransformName = WheelMesh
  wheelType = FREE
  radius = 0.35
  center = 0,0,0
}
```

ModuleWheelSuspension

Defines the spring and damper system that absorbs impacts and controls ride height.

Fields

- **suspensionTransformName:** Transform that visually moves with suspension travel.
- **suspensionDistance:** Maximum vertical suspension travel in meters.
- **targetPosition:** Normalized resting position of the suspension (0–1).
- **springRatio:** Relative stiffness of the suspension spring.
- **damperRatio:** Relative damping strength to control oscillation.

- **useAutoSpring:** Automatically adjusts spring strength based on load.
- **autoSpringDamper:** Automatically adjusts damping when auto spring is enabled.
- **antiRoll:** Transfers load between wheels to reduce body roll.
- **maxLoad:** Maximum supported load before instability or damage risk increases.

Example

```
MODULE
{
  name = ModuleWheelSuspension
  suspensionTransformName = Suspension
  suspensionDistance = 0.45
  targetPosition = 0.6
  springRatio = 8
  damperRatio = 1.2
  useAutoSpring = true
  antiRoll = 0.6
}
```

ModuleWheelSteering

Provides steering by rotating the wheel around a vertical axis.

Fields

- **steeringTransformName:** Transform that rotates for steering.
- **steeringCurve:** Speed-based curve limiting maximum steering angle.
- **steeringResponse:** How quickly the wheel responds to steering input.
- **steeringEnabled:** Whether steering is active by default.

Example

```
MODULE
{
  name = ModuleWheelSteering
  steeringTransformName = Steering
  steeringCurve
  {
    key = 0 25
    key = 10 12
    key = 30 3
  }
}
```

ModuleWheelMotor

Adds powered rotation to wheels, enabling rover propulsion.

Fields

- **maxTorque:** Maximum torque the motor can apply.
- **torqueCurve:** Curve reducing available torque as wheel speed increases.
- **resourceName:** Resource consumed by the motor (typically ElectricCharge).
- **resourceConsumptionRate:** Rate of resource consumption per unit torque.
- **motorEnabled:** Whether the motor is enabled by default.

Example

```
MODULE
{
  name = ModuleWheelMotor
  maxTorque = 3
  resourceName = ElectricCharge
  torqueCurve
  {
    key = 0 3
    key = 10 1.5
    key = 30 0.2
  }
}
```

ModuleWheelBrakes

Controls braking force applied to the wheel when brakes are engaged.

Fields

- **maxBrakeTorque:** Maximum braking torque applied.
- **brakeResponse:** How quickly braking force ramps up.
- **brakeEnabled:** Whether braking is enabled by default.

Example

```
MODULE
{
  name = ModuleWheelBrakes
  maxBrakeTorque = 4
  brakeResponse = 2
}
```

ModuleWheelDeployment

Handles retractable landing gear, including animation and state changes.

Fields

- **animationTrfName:** Transform controlling deployment animation.
- **deployedPosition:** Animation state corresponding to deployed gear.
- **retractedPosition:** Animation state corresponding to retracted gear.

- **retractable:** Whether the wheel can retract.
- **deploySpeed:** Speed of deployment and retraction animation.

Example

```
MODULE
{
   name = ModuleWheelDeployment
   animationTrfName = GearDeploy
   deployedPosition = 1
   retractedPosition = 0
   retractable = true
}
```

ModuleWheelDamage

Simulates structural damage from impacts or excessive loads.

Fields

- **stressTolerance:** Maximum sustained load before damage occurs.
- **impactTolerance:** Maximum impact velocity tolerated without damage.
- **repairable:** Whether the wheel can be repaired by a kerbal.

Example

```
MODULE
{
   name = ModuleWheelDamage
   stressTolerance = 450
   impactTolerance = 300
}
```

Rover Wheel vs Landing Gear Comparison

While both rover wheels and landing gear use the same underlying wheel system, their module composition and tuning differ significantly.

Aspect	Rover Wheel	Landing Gear
Primary Role	Ground propulsion	Takeoff and landing support
ModuleWheelMotor	Required	Not used
ModuleWheelSteering	Often multiple wheels	Usually nose gear only
Suspension	Softer, longer travel	Stiffer, higher load tolerance
Anti-roll	Low to moderate	Moderate to high

Deployment	Not used	Required

Common Wheel Tuning Recipes

This section provides practical, battle-tested tuning patterns for common use cases. Values are starting points and should be adjusted based on mass, gravity, and wheel count.

Light Rover (Mun / Minmus)

- **suspensionDistance:** 0.2 – 0.3 (enough travel for terrain irregularities)
- **targetPosition:** 0.45 – 0.55 (neutral ride height)
- **springRatio:** 3 – 4 (soft suspension)
- **damperRatio:** 0.9 – 1.1 (prevents bounce without sluggishness)
- **antiRoll:** 0 – 0.3 (avoid wheel lifting on uneven ground)
- **maxTorque:** Low to moderate; rely on torqueCurve

Example:

```
MODULE
{
   name = ModuleWheelSuspension
   suspensionDistance = 0.25
   targetPosition = 0.5
   springRatio = 3.5
   damperRatio = 1.0
}
```

Heavy Rover (Duna / Eve)

- **suspensionDistance:** 0.3 – 0.45 (greater mass absorption)
- **targetPosition:** 0.6 – 0.7 (pre-compressed under load)
- **springRatio:** 5 – 7 (prevent bottoming out)
- **damperRatio:** 1.1 – 1.3
- **antiRoll:** 0.3 – 0.6 (stability during turns)
- **useAutoSpring:** Strongly recommended

Example:

```
MODULE
{
   name = ModuleWheelSuspension
   suspensionDistance = 0.4
   targetPosition = 0.65
   springRatio = 6
   damperRatio = 1.2
   useAutoSpring = true
```

```
  antiRoll = 0.5
}
```

Aircraft Main Landing Gear

- **suspensionDistance:** 0.4 – 0.6 (absorb touchdown energy)
- **targetPosition:** 0.65 – 0.75 (sit low at rest)
- **springRatio:** 8 – 12
- **damperRatio:** 1.2 – 1.5 (prevent bounce on rollout)
- **antiRoll:** 0.6 – 0.9
- **maxBrakeTorque:** High; match expected landing mass

Example:

```
MODULE
{
  name = ModuleWheelSuspension
  suspensionDistance = 0.5
  targetPosition = 0.7
  springRatio = 10
  damperRatio = 1.4
  antiRoll = 0.8
}
```

Nose Landing Gear

- **suspensionDistance:** Slightly less than mains
- **targetPosition:** 0.55 – 0.65
- **springRatio:** Slightly softer than mains
- **damperRatio:** Match main gear
- **antiRoll:** Lower than main gear
- **steeringCurve:** Aggressively limit at speed

Example:

```
MODULE
{
  name = ModuleWheelSuspension
  suspensionDistance = 0.35
  targetPosition = 0.6
  springRatio = 7
  damperRatio = 1.3
}
```

Hard-Landing Spaceplane / SSTO

- **suspensionDistance:** Maximum practical
- **targetPosition:** 0.75 – 0.85

- **springRatio:** High, but avoid rigidity
- **damperRatio:** High to kill rebound
- **impactTolerance:** Increase in ModuleWheelDamage

Example:

```
MODULE
{
  name = ModuleWheelSuspension
  suspensionDistance = 0.6
  targetPosition = 0.8
  springRatio = 11
  damperRatio = 1.5
}
```

Troubleshooting Decision Trees

Use these decision trees when a wheel setup behaves poorly. Start from the symptom you see in-game, then apply the suggested adjustments in the listed order. Make one change at a time and re-test.

Symptom: Wheel bounces / oscillates repeatedly

1. **Step 1:** Increase damperRatio by +0.1 to +0.3 until oscillation stops.
2. **Step 2:** If it still bounces, reduce springRatio slightly (−5% to −15%).
3. **Step 3:** If bounce happens only when mass changes (cargo/fuel), enable useAutoSpring and autoSpringDamper.
4. **Step 4:** If bounce happens after touchdown, increase brakeResponse slightly or reduce maxBrakeTorque to avoid skid-hop feedback.

Notes: Root causes: insufficient damping, overly stiff spring for the actual load, or mass shifts mid-run.

Symptom: Suspension bottoms out (wheel compresses fully)

5. **Step 1:** Increase springRatio (+10% to +30%).
6. **Step 2:** Increase suspensionDistance (+0.05 to +0.15 m) if model allows travel.
7. **Step 3:** Increase targetPosition (e.g., 0.55 → 0.65) to start more compressed under load.
8. **Step 4:** Enable useAutoSpring for variable-mass craft (recommended).

Notes: Root causes: spring too soft, insufficient travel, or target position too extended for the load.

Symptom: Craft ‘pops’ upward / wheel feels too rigid on contact

9. **Step 1:** Lower springRatio (−10% to −25%).
10. **Step 2:** Lower targetPosition (e.g., 0.7 → 0.55) so it doesn’t start over-compressed.
11. **Step 3:** If popping happens only at high speed, increase damperRatio slightly (+0.1 to +0.2).

Notes: Root causes: overly stiff spring or starting too compressed, causing rebound energy.

Symptom: Rover flips when turning at speed

12. **Step 1:** Reduce steering at speed via steeringCurve (lower the angle above ~10–15 m/s).
13. **Step 2:** Increase antiRoll moderately (+0.1 to +0.3).
14. **Step 3:** Lower center of mass (craft design), widen wheelbase, or reduce ride height (targetPosition up slightly).
15. **Step 4:** Reduce maxTorque and/or make torqueCurve drop faster with speed.

Notes: Root causes: excessive steering angle at speed, high CG, insufficient anti-roll, or too much torque.

Symptom: Wheel slips / can't climb / spins out easily

16. **Step 1:** Reduce maxTorque or shape torqueCurve for lower torque at low speed if it's just wheelspin.
17. **Step 2:** If it lacks power instead, increase maxTorque slightly and add a more gradual torqueCurve drop-off.
18. **Step 3:** Increase frictionMultiplier in ModuleWheelBase (small increments, e.g., 1.0 → 1.1).
19. **Step 4:** Ensure enough driven wheels and manage mass distribution.

Notes: Root causes: torque too high for available traction, friction too low, or insufficient driven contact.

Symptom: Nose gear shimmy / wobble on rollout

20. **Step 1:** Increase damperRatio slightly (+0.1 to +0.2).
21. **Step 2:** Reduce steeringResponse and/or reduce steeringCurve angles at speed.
22. **Step 3:** Reduce maxBrakeTorque on the nose gear (brakes on nose can induce oscillations).
23. **Step 4:** Increase antiRoll slightly on the main gear (stabilizes body roll that feeds shimmy).

Notes: Root causes: over-responsive steering, under-damped suspension, or braking-induced oscillation.

Symptom: Retractable gear behaves oddly (collides, twists, or 'ghost' contact)

24. **Step 1:** Verify animationTrfName and gear animation axes match expected direction.
25. **Step 2:** Ensure wheelColliderTransformName is correctly positioned for deployed state.
26. **Step 3:** Check that steeringTransformName and suspensionTransformName are not parented under moving transforms incorrectly.
27. **Step 4:** If contact persists while retracted, verify deployment module is correctly disabling wheel interaction.

Notes: Root causes: transform hierarchy issues and mismatched collider/animation setup.

Runtime Diagnostics Code (C# – KSP 1.x)

The snippets below are intended for mod/plugin diagnostics. They show how to discover wheel modules on a vessel and print useful state information (grounded, broken, steering/motor enabled, etc.). Exact property names can vary by KSP minor version; treat these as practical templates and adjust after checking the API in your target version.

1) Enumerate wheel modules on the active vessel

```
// KSP 1.x – enumerate wheels on the active vessel
using System.Linq;
using UnityEngine;

public static class WheelDiagnostics
{
    public static void DumpActiveVesselWheels()
    {
        var v = FlightGlobals.ActiveVessel;
        if (v == null)
        {
            Debug.Log("[WheelDiag] No active vessel.");
            return;
        }

        var bases = v.FindPartModulesImplementing<ModuleWheelBase>();
        Debug.Log($"[WheelDiag] Vessel '{v.vesselName}' wheels: {bases.Count}");

        foreach (var wb in bases)
        {
            var p = wb.part;
            // Common high-value state
            bool grounded = wb.isGrounded;
            bool broken   = wb.isBroken;

            // Optional associated modules
            var sus   = p.FindModuleImplementing<ModuleWheelSuspension>();
            var steer = p.FindModuleImplementing<ModuleWheelSteering>();
            var motor = p.FindModuleImplementing<ModuleWheelMotor>();
            var brake = p.FindModuleImplementing<ModuleWheelBrakes>();
            var dep   = p.FindModuleImplementing<ModuleWheelDeployment>();
            var dmg   = p.FindModuleImplementing<ModuleWheelDamage>();

            Debug.Log(
                $"[WheelDiag] Part={p.partInfo?.name ?? p.name} " +
                $"Grounded={grounded} Broken={broken} " +
                $"Susp={(sus!=null)} Steer={(steer!=null)} Motor={(motor!=null)}
Brake={(brake!=null)} " +
                $"Deploy={(dep!=null)} Damage={(dmg!=null)}"
            );
        }
    }
}
```

2) Print suspension tuning values (from the part's modules)

```
// Print suspension-related tuning values for each wheel
public static void DumpSuspensionTuning(Vessel v)
{
    if (v == null) return;

    var bases = v.FindPartModulesImplementing<ModuleWheelBase>();
    foreach (var wb in bases)
    {
        var p = wb.part;
        var sus = p.FindModuleImplementing<ModuleWheelSuspension>();
        if (sus == null) continue;

        // These fields are commonly present in configs; adjust names if your target version differs
        Debug.Log(
            $"[WheelDiag] {p.partInfo?.name ?? p.name} " +
            $"suspDist={sus.suspensionDistance:F3} " +
            $"targetPos={sus.targetPosition:F2} " +
            $"springRatio={sus.springRatio:F2} " +
            $"damperRatio={sus.damperRatio:F2} " +
            $"antiRoll={sus.antiRoll:F2} " +
            $"autoSpring={sus.useAutoSpring} autoDamper={sus.autoSpringDamper}"
        );
    }
}
```

3) Print steering and motor states

```
// Print steering curve sampling and motor limits
public static void DumpSteerMotor(Vessel v, float sampleSpeedMps = 15f)
{
    if (v == null) return;

    var bases = v.FindPartModulesImplementing<ModuleWheelBase>();
    foreach (var wb in bases)
    {
        var p = wb.part;
        var steer = p.FindModuleImplementing<ModuleWheelSteering>();
        var motor = p.FindModuleImplementing<ModuleWheelMotor>();

        if (steer != null)
        {
            // Many KSP wheel modules use FloatCurve-like curves; naming may differ
            float maxSteerAtSample = steer.steeringCurve.Evaluate(sampleSpeedMps);
            Debug.Log($"[WheelDiag] {p.partInfo?.name ?? p.name} steer@{sampleSpeedMps:F0}m/s = {maxSteerAtSample:F1} deg");
        }

        if (motor != null)
        {
            Debug.Log($"[WheelDiag] {p.partInfo?.name ?? p.name} maxTorque={motor.maxTorque:F2} resource={motor.resourceName}");
        }
```

```
    }
}
```

4) Minimal ModuleManager patch pattern for safe iterative tuning

```
// Example ModuleManager patch to iteratively tune suspension on a wheel part
// Replace part name with the wheel you are testing
@PART[YourWheelPartNameHere]
{
    @MODULE[ModuleWheelSuspension]
    {
        @suspensionDistance = 0.35
        @targetPosition = 0.60
        @springRatio = 6.0
        @damperRatio = 1.2
        %useAutoSpring = true
        %autoSpringDamper = true
        %antiRoll = 0.4
    }
}
```

Tip: for diagnostics, you can also dump the raw PartModule fields via reflection if a property name differs in your target version. Start by logging all fields/properties on the module type, then map them to the config names you care about.