# Configuring a Physics Body

Move a physics body, and make it collide with other objects, by setting its physical properties once or changing them dynamically.

## Overview

You configure mass, area and density to values that create the right level of momentum and transfer of kinetic energy upon collisions of your in-game objects. Though you typically configure a physics body only once, there are exceptions; for example, when a node crosses into a different environment, such as land to space, or land to water.

## Use Mass to Resist Acceleration

You set the mass on every volume-based body in your scene so that it properly reacts to forces applied to it.

A physics body's mass, area, and density properties are all interrelated. When you first create a body, the body's area is calculated, and never changes afterward. The other two properties change values at the same time, based on the following formula:

mass = density x area

When you configure a physics body, you have two options:

>> Set the mass property of the body. The density property is then automatically recalculated. This approach is most useful when you want to precisely control each body's mass.

>> Set the density property of the body. The mass property is then automatically recalculated. This approach is most useful when you have a collection of similar bodies created with different sizes. For example, if your physics bodies were used to simulate asteroids, you might give all asteroids the same density, and then set an appropriate bounding polygon for each. Each body automatically computes an appropriate mass based on its size on the screen.

## Time Your Changes to a Physics Body's Properties

Most often, you configure a physics body once and then never change it. For example, the mass of a body is unlikely to change during play. However, you are not restricted from changing it. Some kinds of games may require the ability to adjust a body's properties even while the simulation is executing. Here are a few examples of when you might do so:

>> In a realistic rocket simulation, the rocket expends fuel to apply thrust. As fuel is used up, the mass of the rocket changes. To implement this in SpriteKit, you might create a rocket class that includes a fuel property. When the rocket thrusts, the fuel is reduced and the corresponding body's mass is recalculated.

>> The damping properties are usually based on the body's characteristics and the medium it is traveling through. For example, a vacuum applies no damping forces, and water applies more damping forces than air. If your game simulates multiple environments and bodies can move between those environments, your game can update a body's damping properties whenever it enters a new environment.

>> When a contact delegate responds to a collision, and an in-game object, such as a rocket, should be destroyed, you might set the rocket node's `physicsBody` to nil. This prevents further contact delegate callbacks for the rocket as it plays out its final explosion animation before being deallocated.

Typically, you make these changes as part of scene pre- and post-processing, using the `SKSceneDelegate` methods.