// Lesson 4 - Joints - Connecting Physical Bodies

> What you will learn:

- The use of physical joints to connect bodies in a way to create different behaviours
- Types of joints and the underlying principles
- Constructing specific Joints, Spring

> Requirements Knowledge/Files:

- Knowledge how to setup a (physics) scene in Fudge
- Knowledge of the basic Component System of Fudge
- Hint! - You can use the physics boilerplate in the tutorial folder

> TL;DR; - But it's worth to read the long text, to get detailed infos and deeper understanding

- Physical objects ignore parenting of normal Fudge Nodes, they need to be connected by joints or holding each other
- Joints are Components and are used like any other Fudge Component with the difference that all of them need a ComponentRigidbody that they are attached to and a second that they connect with on a anchorpoint
- Each joint type has different properties and construction conditions
- Joints generally consist of a translationalMotor that defines forces, limits, of movement along a defined axis of freedom, or a rotationalMotor that does the same for rotations. And lastly springs that can be streched or compressed. Joints can have multiple of those elements or none.

> Step 1 - What's a joint

A joint is a connecting piece between to physical object that is defining the behaviour these two bodies have with each other. It's similar to the transform hierarchy but also different. Transforms have only one type of connection, parent/child resulting in the child always following what the parent is doing. Physical bodies are the opposite they do nothing else but their own stuff and only react to constraints physical rules apply on to them. E.g. a physical body is falling, unless some obstacle is in the way, which is the application of the physical rule of collision and gravity.

So we need a way to connect bodies to have a similar behaviour than parent/child concepts. Luckily the real world has a thing for this, joints. A joint can be screw, nail, glue, anything that is connecting bodies and having defined rules of how the two bodies are now connected.

Joints can have different names and usages, that's why there are different types of joints in Fudge to cover the most common use cases. In other engines you might find something like a generic joint that you need to configure fully by yourself or can choose presets. But you can always find some basic joints like the prismatic joint, revolute (hinge) joint and universal joint. Now those names do not say much and we will come back to that in joint types. For now in short, behind these names stands a decision. How much freedom does my connection have?

A door is connected to a door frame, but it's still able to be opened, by swinging. That's a physical degree of freedom, the door can swing on one axis, or degree. This is also an example of a revolute joint, also known as hinge. More degrees could be that it's can not only swing but also move.

So choosing a type of joint means choosing the amount of degrees of freedom and what kind of freedom, either the freedom to rotate or move.

> Step 2 - Usage and manipulation

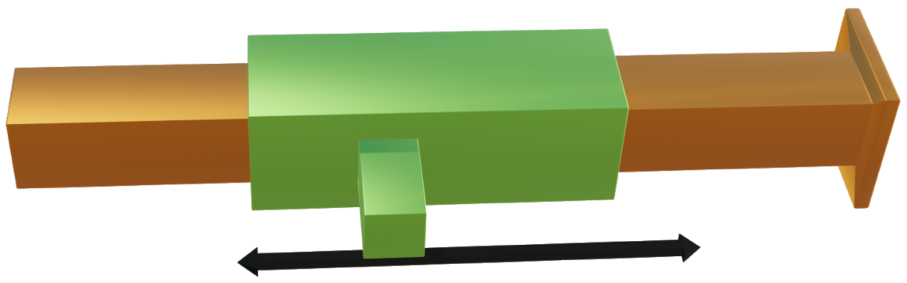
Joints are physics components and need to be attached to a node like any other component therefore the usage is similar:

```
let joint : f.ComponentJointTypeName = new
f.ComponentJointTypeName(attachedRigidbody, connectedRigidbody, axis,
anchorLocalFromAttachedRB);
node.addComponent(joint);
```

Since a joint is connecting two bodies you have to define the two bodies when creating the joint component. The attachedRigidbody is the body of the node where you want to add the component. Back to our door example, the attachedRigidbody is the doorframe, since we want to connect the door to our frame. Following this logic the connectedRigidbody is the other body, or door in this case. Now we want the joint to swing open/close therefore our axis would be f.Vector3(0,0,1) and the anchor is at the base of the doorframe. We then add this newly created component to our door frame and we have door which open and closes when a rigidbody walks against it.
This is highly simplified but that's the basic concept of joints, that we are continuing on.

> Step 3 -Types of joints, degrees of freedom

Now we built a revolute joint so far, a door hinge. This is the most basic joint in terms of rotation, we have one degree of freedom, and a axis where it swings on around a anchor.
But let's take a step back and go to the most basic form of a joint, a **prismatic joint**. Often used to create a slider, or a spring. You have something like this in every pen. A prismatic joint means, 1 degree of freedom, in movement on one axis.



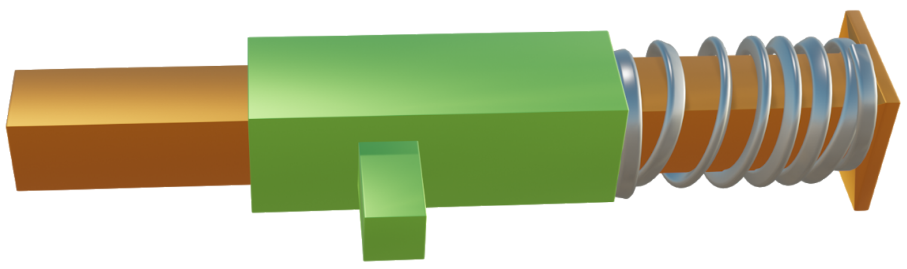
Simple, two bodies, one slides along the other, it can not move in any other direction because it’s slid onto the other. You can move it along from one end to the other, and not further (most of the times). This is what’s called a motor, a motor defines the behaviour of movement/rotation. A motor consists of:

- Limits: Upper, Lower to define how far it can move, if you define it as 0/0 the second body and the first body can’t move away from one another, which is like glued together, or screwed. The physics engine will try to hold the two bodies together, but with less accuracy this can result in some jiggling.

- Force / Torque: The power of the connection holding. While the limits are set in place and the body can’t move out of them (normally) you can define a force that makes it harder to slide for example. This force is also used to achieve the defined speed, if one is defined.

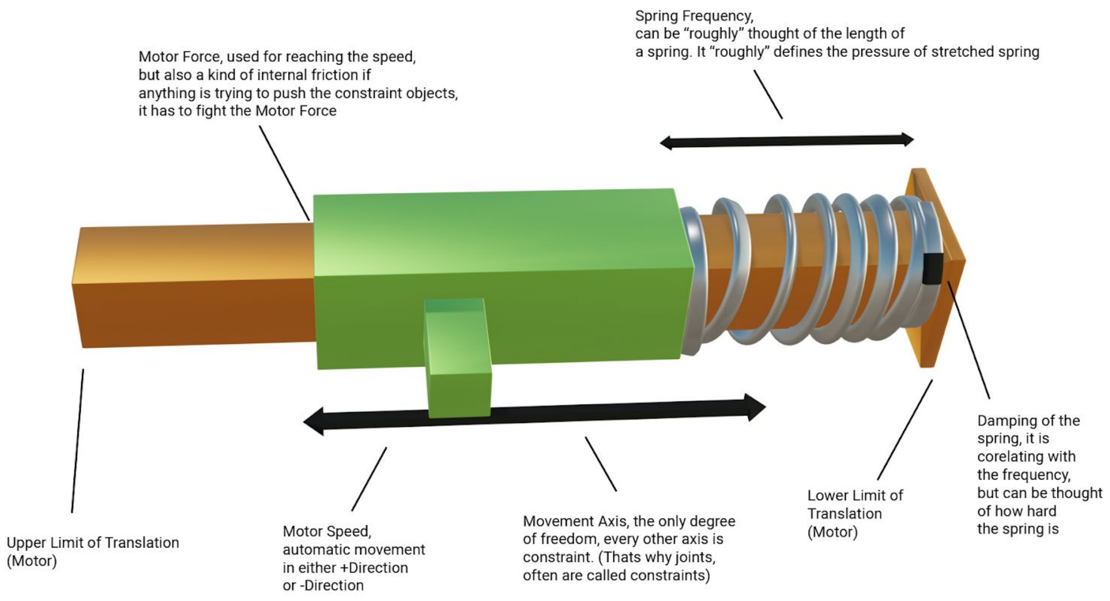
- Speed: You can have your joint move by itself. Define a speed to for example slide the green body to the right by setting a minus speed or to the left limit by plus speed. This can naturally also achieved by applying forces to the green body in the example.

So this is a motor, a motor can be set for either movement or rotation, or multiple depending on the joint type. The default is always that the limits are -10/10 meters for movement and 0-360° degree for rotation, so newly created joints aren't restricting much.
But there is more a joint can also have a spring feature, deping on the type. Meaning there is a build in mechanism that is making it possible to go over the set motor-limits by stretching or compressing, but also trying to reset the position.



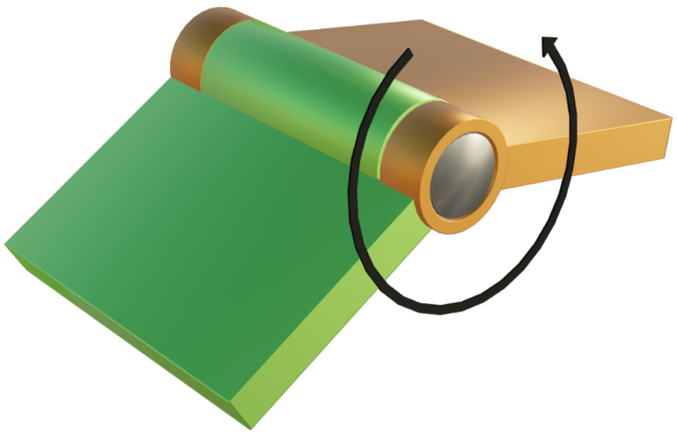
Now a spring has two settings, a frequency and a damping. It's a complex physical phenomena but basically it's controlling the stiffness of the spring and the amount of stretch and stretching counterforce (or speed). If the frequency is 0 there is no spring, that's always the default setting for joints with springs, so there is no spring active, we will look at creating a spring in the example later.

But this concludes the actual basic structure of joints. (Visual - **Prismatic Joint**)

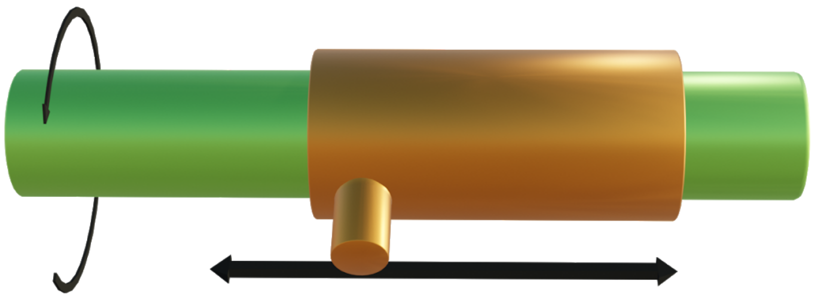

Motor Force, used for reaching the speed, but also a kind of internal friction if anything is trying to push the constraint objects, it has to fight the Motor Force
Spring Frequency, can be "roughly" thought of the length of a spring. It "roughly" defines the pressure of stretched spring
Damping of the spring, it is corelating with the frequency, but can be thought of how hard the spring is
Lower Limit of Translation (Motor)
Movement Axis, the only degree of freedom, every other axis is constraint. (Thats why joints, often are called constraints)
Motor Speed, automatic movement in either +Direction or -Direction
Upper Limit of Translation (Motor)

There are more types of joints which i will only name and visualize here, since they function similar you should be able to use them on your own.

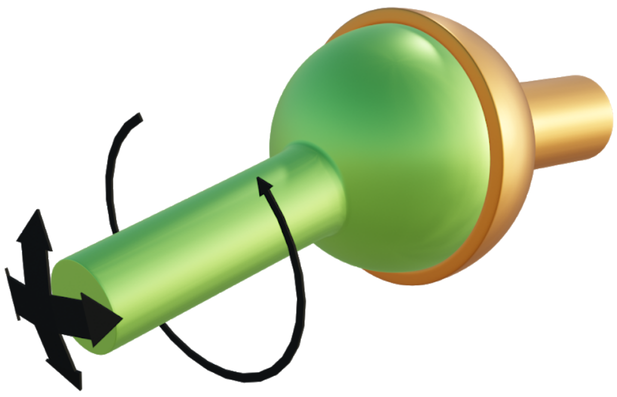
Revolute Joint: (often called Hinge)



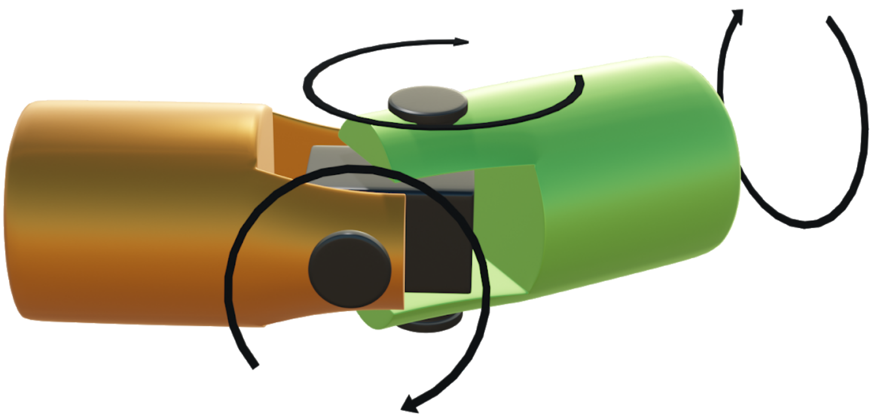
Cylindrical Joint: (pretty much the combination of hinge + prismatic)



Sphere Joint: (also known as Ball And Socket joint, your shoulder is simplified a sphere joint)



Universal Joint: (often used in cars, because they rotate somewhat independent but transfer the twist)



Ragdoll Joint: This joint does not need a visualization, it's a more complex type of sphere joint specialized for creating ragdolls. A game concept of creating a limb body that can represent a creature falling. You will rarely need it and setting it up is a hassle because you have to test your limbs constantly to have a realistic falling body, but it's there if needed.

> Step 4 - Practical example, creating a damping spring

A simple spring damping. We heard of a prismatic joint, it can move in one direction and has a spring element. So we create a prismatic joint between two cubes. The cubes you should know how to create.

```
//Cube creation - Similar to lesson 1
let holderBody = createCompleteNode("JointHolder_Prismatic",
fixedJointMaterial, new f.MeshCube(), 0, f.PHYSICS_TYPE.STATIC);

holderBody.mtxLocal.translate(new f.Vector3(-2, 1, 0));
hierarchy.appendChild(holderBody);

let springboardBody = createCompleteNode("Connected_Prismatic",
jointMaterial, new f.MeshCube(), 1, f.PHYSICS_TYPE.DYNAMIC);

springboardBody.mtxLocal.translate(new f.Vector3(-2, 3, 0));
springboardBody.mtxLocal.scale(new f.Vector3(1.5, 0.2, 1.5));
hierarchy.appendChild(springboardBody);

//Construction of a prismatic joint(body, body, axis, anchor)
let prismaticJoint = new
f.ComponentJointPrismatic(holderBody.getComponent(f.ComponentRigid
body),
springboardBody.getComponent(f.ComponentRigidbody),
new f.Vector3(0, 1, 0), new f.Vector3(0, 0.5, 0));

holderBody.addComponent(prismaticJoint);
```

Create a third cube above the first to, so it comes down on our spring.

After that we only need to give our joint the correct settings to make it feel like a simple damping spring.

```
prismaticJoint.motorLimitUpper = 0;
prismaticJoint.motorLimitLower = 0;
prismaticJoint.springDamping = 0.2;
prismaticJoint.springFrequency = 0.7;
```

Now you will notice that when throwing a heavy cube onto the spring it does go through the holderBody, because joint bodies do not collide with each other by default. This is wanted behaviour so you can build a slider out of two cubes, where one is slid onto the other, that would normally not work since the second cube has no hole where he can be slid onto anything but since they do not collide with each other it's no problem.

```
 prismaticJoint.internalCollision = true;
```

Now since joints are realistic mechanical parts, so you are able to break them, if you want to. It's most of the times not necessary, but think of a game where you want to walk over a bridge, but when carrying too much weight it breaks. By default joints are unbreakable but you can define breakForce and breakTorque for movement and rotation if needed.

```
prismaticJoint.breakForce = 20;
```

You might experience weird behaviour with joints, especially if you connect a few of them together in this case go on to Lesson 1, or play with physics settings by yourself. The defaults are always a less accurate version in favor of performance.

Hint! - This concludes this tutorial but keep in mind not everything is explained in detail you can explore things better by yourself! Detailed examples can are found in the finished tutorial folder and all joint types are used in the miscellaneous/experiments/marko folder of fudge