# Welcome to the Fidget wiki!

You are probably wondering “Why is your wiki a google doc?”. The answer is quite simple my friend! I just didn’t have anything better laying around and didn’t feel like searching too much.

Let's start, shall we?

## FAQ:

When I download the library, it errors.
Well the folder you should have in your directory is “Fidget”
When you download from github it gives you a zip and in that zip is a folder with the branch name called “fidget-main” you need to take just the “Fidget” folder out of that and put it in your avatar

### Current version: v1.0

## Getting started:

To begin using fidget, you need to require it like so:

```
local fidget = require("Fidget.FidgetSetup")
```

After that your first function to run(after entity init) is:

```
local rigidbody = fidget.rigidbodies.createRigidbody({pos =
player:getPos()})
```

This will create a rigidbody with default parameters at your location.

After this lets move on to joints. Currently there is only one joint type with more planned in the future.

To create a joint you first need 2 rigidbodies. To connect them with a joint you need to call the appropriate function
(The code is on the next page cause it doesnt fit)

```
  local rigidbody1 = fidget.rigidbodies.createRigidbody({pos =
player:getPos()})
  local rigidbody2 = fidget.rigidbodies.createRigidbody({pos =
player:getPos()})
    fidget.joints.createJoint(
    {
      rigidbody1 = rigidbody1,
      rigidbody2 = rigidbody2,
    }
    )
```

Congrats! Now you have 2 rigidbodies that fall down and interact with the world! Though there is a bit of a problem. They just fall down when you spawn them. Let's give them some velocity on spawn.

```
  local rigidbody1 = fidget.rigidbodies.createRigidbody({pos =
player:getPos(),vel = player:getLookDir()*10})
  local rigidbody2 = fidget.rigidbodies.createRigidbody({pos =
player:getPos(),vel = player:getLookDir()*10})
    fidget.joints.createJoint(
    {
      rigidbody1 = rigidbody1,
      rigidbody2 = rigidbody2,
      distance = 1,
    }
    )
```

How nice.

You now know the absolute basics. Let’s move onto the documentation

## Rigidbodies function:

```
fidget.rigidbodies.createRigidbody(params<table>)
```

This creates a rigidbody with the given parameters, these parameters include:

```
pos - position(vec3)
vel - velocity(vec3)
rot - rotation(vec4<quaternion>)
rotVel - angular velocity(vec3)
mass - mass(scalar)
dimensions - dimensions of the cuboid(full dimensions not half
extents)(vec3)
gravity - do i really have to explain this(vec3)
friction - coefficient of friction(scalar)
linearMovement - contrary to the name it makes a body unable to move in
any direction(bool)
model - modelpart that the body should appear as(modelpart)
modelScale - scales the modelpart(vec3)
worldCollision - controls if the body collides with the world(bool)
isSleeping - controls if the body is sleeping on spawn(bool)
```

All these values are read/write and the fastest both in terms of instructions and ms is to just index the rigidbody directly. If you want to use functions to edit them they are below.

## Rigidbody methods:

```
rigidbody:remove()
```

Deletes a rigidbody

```
rigidbody:addForce(force(vec3))
```

Adds force to the rigidbody(only linear movement)

```
rigidbody:addForceAtPoint(point(vec3), force(vec3))
```

Adds force to the rigidbody at a point.

```
rigidbody:setPos(pos(vec3))
```

Sets the rigidbody position to the given vec3.
Same function exists for:
**vel,rot,rotVel,mass,dimensions,gravity,friction,linearMovement,modelScale,worldCollision,isSleeping**
(im to lazy to write that all out)

```
rigidbody:getPos()
```

Gets the rigidbody position.
Same function exists for:
**vel,rot,rotVel,mass,dimensions,gravity,friction,linearMovement,modelScale,worldCollision,isSleeping**

## Physics Simulation

Ahh the variables in here control the simulation(duh). You can control the quality, some other parameters and turn on debug visuals.

There are no functions this time around though(I'm too lazy to make them).

I'm feeling kinda lazy so I'm just gonna put part of the code here

```
physicsSim.physicsIterations = 2 --physics steps per tick
physicsSim.dt = (1/20)/physicsSim.physicsIterations --delta time controls
how fast the simulation runs if its lower then the simulation runs slower
physicsSim.step --step count(how many steps happened since the physics
started)
physicsSim.velocityIterations = 4 -- improves stability and convergence
physicsSim.baumgarteMultiplier = 0.2 --what is this german-ass name bruh.
Baumgarte? more like "Ba! En garde!" controls how violently rigidbodies
separate
physicsSim.slop = 0.000 -- sloppy,sloppy, little slop. How to slop slop
slop. Never gonna slop you up never gonna slop you down and slop around
and slopert you! but fr this is slop for baumgerte
physicsSim.relaxation = 1 --mightTM improve stability and convergence(but
from my testing it didnt lmao)
physicsSim.cacheMultiplier = 0.9--How much of the cached impulse comes
over to the next physics step
physicsSim.broadPhaseCollision = "aabb" --options: sphere, aabb controls
how broadphase is conducted, aabb is faster
physicsSim.normalSnappingThreshold = 0.01--for stability, dont go too high
on this one
physicsSim.sleeping = false-- if true rigidbodies can eep. eepy time. I
sleep too. What does it mean to sleep? how many years have passed since i
went into bed(probably around 3/(365*2))
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
physicsSim.sleepTimeThreshold = 0.1 --how long does a body need to be
still to sleep in seconds
physicsSim.sleepVelocityThreshold = 0.1 -- How fast does a body need to
move to sleep
physicsSim.sleepRotVelocityThreshold = 0.02
```

```
physicsSim.waterDensity = 1    -- controls buoyancy
physicsSim.waterDamping = 0.92 -- how much is the velocity and ang
velocity slown down in water
physicsSim.contactPointMergingThreshold = 0.03--how close do pointy points
need to be to be one pointy point
physicsSim.solver = "regularPGS" --either regularPGS or
scuffedSplitImpulse the 1st one has more accurate friction but is less
stable with stacking, the 2nd is way more stable but has a few "bugs" with
friction sometimes. generally option 2 is better except for accuracy
critical things like dominoes
physicsSim.debug = {
worldColliders = false,  --shows the world colliders
contacts = false,    --shows the contact points with impulses applied
axis = false, --shows axis of the body at local origin of the body
broadphaseAABB = false, --shows aabbs used in broadphase collision
collisionNormals = false, --show the normals at collisions(the the local
space origin of one of the bodies)
waterVolumes = false, --shows the volumes used for buoyancy
joints = true, --shows the joints
}
```

(the humor does not hit)
To edit any of the values you just do:

```
fidget.physicsSim.<index>
```

Or in case of the debug values:

```
fidget.physicsSim.debug.<index>
```

——----------

There also is one function for changing the simulation quality:

```
fidget.physicsSim.changeQuality(quality<string>)
```

For the quality you have 4 options: “high”,”medium”,”low”,”lowest”

That’s all

## Joints

Finally the end is coming

```
joints.createJoint(params)
```

Creates a joint with the given parameters

Those parameters include:

```
    pos1 - the position of the first end of the joint in local space of
the first body
    pos2 - the position of the first end of the joint in local space of
the second body
    rigidbody1 - no need to explain
    rigidbody2 - no need to explain AGAIN
    distance - the distance 2 bodies will keep from eachother
```

Gotta love the formatting.

Anyways there are functions for setting/getting the pos1,pos2,distance just like the rigidbodies(eg. joint:setPos1() ) and just like with rigidbodies it's faster to edit them directly.

The end

This is not my real email btw