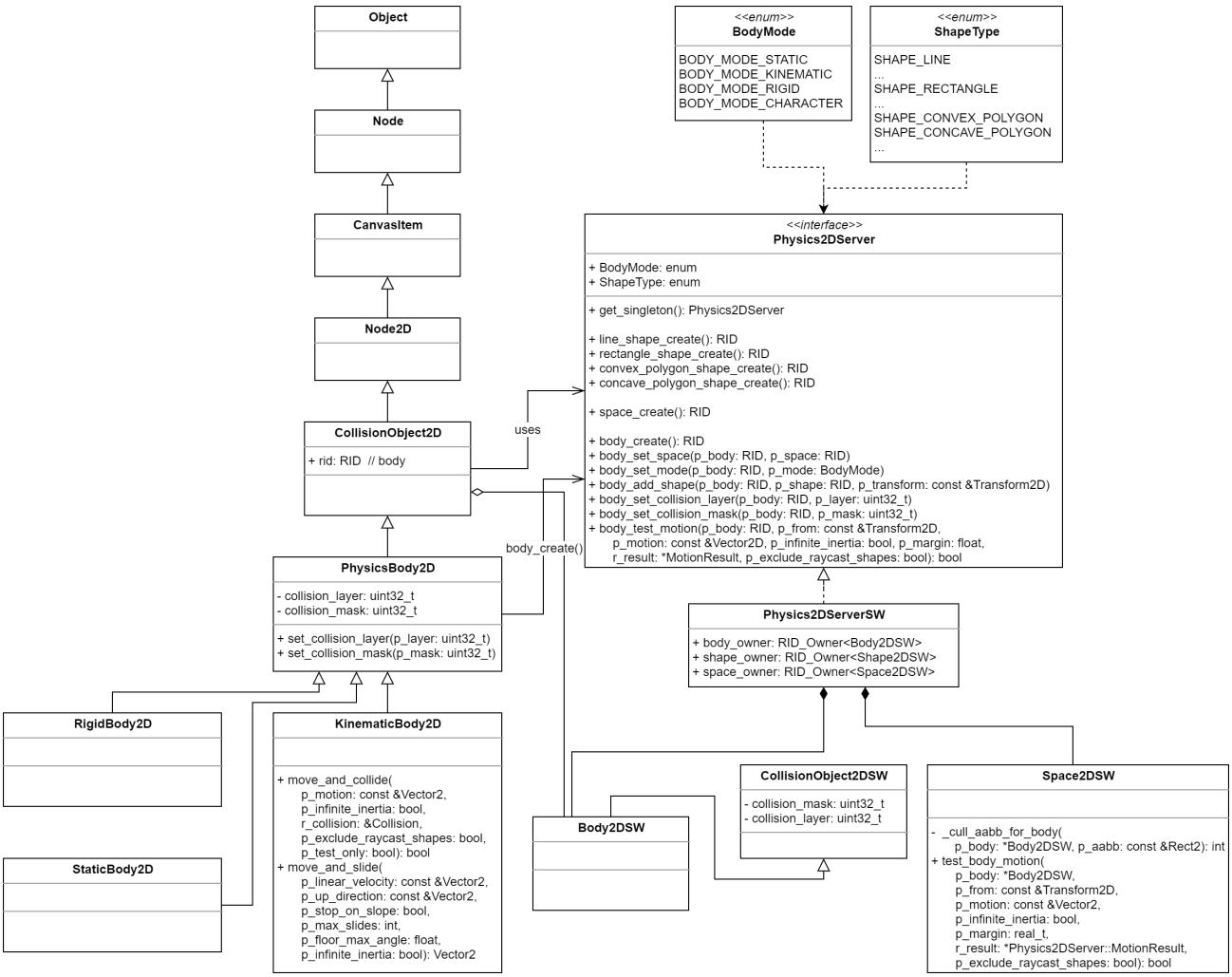
Godot Physics 2D overview

Physics2D classes



`Physics2DServer`

Central API interface for the 2D physics subsystem. Provides the functionality for creating bodies, spaces and shapes and configuring and linking them.

The implementors (such as `Physics2DServerSW`) have their own type hierarchy, which is opaque.

The main users of this class are the scene classes, such as `CollisionObject2D`, `KinematicBody2D` and others. The user classes do not have any knowledge of the actual `Physics2DServer` implementation, and all instances of various underlying physics entities such as `Body2DSW` are hidden by the use of `RID`s, which are passed around.

The actual implementations of `Physics2DServer` own all the various physics objects instances and map them to `RID`s.

`CollisionObject2D`

Basic scene node class for 2D physics objects. Holds the `RID` handler to the physics object of the actual underlying physics server and provides some basic functionality for shape management, common for bodies and areas.

`PhysicsBody2D`

Scene node class for 2D physics bodies, base class for specializations such as rigid, kinematic and static bodies.

`KinematicBody2D`

Non-simulated physical body, which is driven by user code. When using the `Physics2DServerSW` implementation, references a `Body2DSW` with `BodyMode::BODY_MODE_KINEMATIC` mode set.

`Physics2DServerSW`

Godot reference implementation for the 2D physics server.

Functioning

`KinematicBody2D::move_and_collide()`

The method is available to scripts via GDNative, and when called, proxies the call to the underlying `PhysicsServer2D` implementation.

In `PhysicsServer2DSW` implementation, the call is proxied to `Space2DSW::test_body_motion()`, which roughly does the following:

1. Merge the AABB of all shapes into enclosing box.
2. Apply any given margin to it.
3. Try to free the body if it's stuck, before trying to apply any motion:
    i. Get the collision candidates via `Space2DSW::_cull_aabb_for_body()`.

    NOTE: this function also does the collision mask checks, excludes exceptions, etc.

    ii. Check for intersections of each of body's shapes with the collision candidates' shapes. Here all sort of checks are done, for example, the one-way collisions are skipped if necessary and so on.

    iii. If any collisions are found, attempt to compute the recover motion and apply it to body's transform. This is some kind reverse motion vector, derived by the sum of penetration vectors.

    iv. Repeat from step 1 for a number of times.
4. Perform the actual motion.