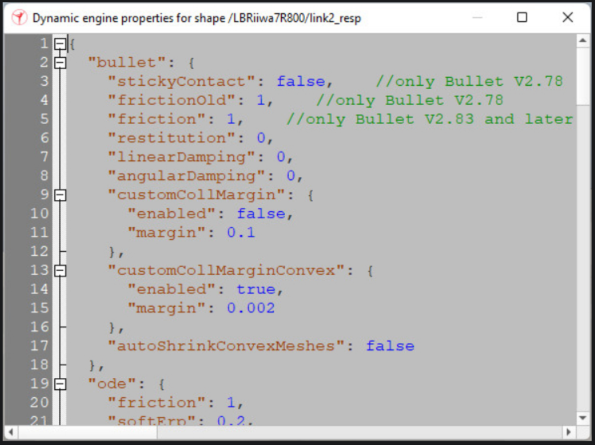
Physics engine differences

Each one of the physics engine supported in coppeliaSim operates in a different manner, and each one of them will generate slightly or completely different results. Results depend on the engine's specific algorithms and calculation routines, but also on the engine specific properties/parameters used. While it would be possible to very roughly abstract some individual properties (e.g. the coefficient of friction for a body), the result would often not be the one expected: there is rarely a one-to-one correspondency between engine properties/parameters, and for this reason, the user can adjust them on an engine-basis. The default engine properties/parameters have been selected to perform best in most situations, but depending on the task and application, one will often have to fine-tune them for stability or realism. This is done for dyn. properties of shapes, dyn. properties of joints and global engine properties, via a JSON editor:

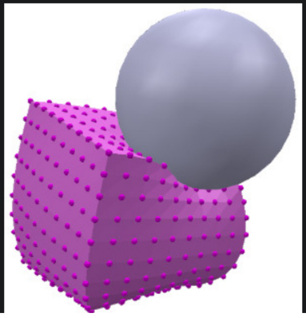

Dynamic engine properties for shape /LBRiiwa7R800/link2_resp

```
{
  "bullet": {
    "stickyContact": false,    //only Bullet V2.78
    "frictionOld": 1,    //only Bullet V2.78
    "friction": 1,    //only Bullet V2.83 and later
    "restitution": 0,
    "linearDamping": 0,
    "angularDamping": 0,
    "customCollMargin": {
      "enabled": false,
      "margin": 0.1
    },
    "customCollMarginConvex": {
      "enabled": true,
      "margin": 0.002
    },
    "autoShrinkConvexMeshes": false
  },
  "ode": {
    "friction": 1,
    "softErp": 0.2,
```

[Engine properties editor, for a shape]

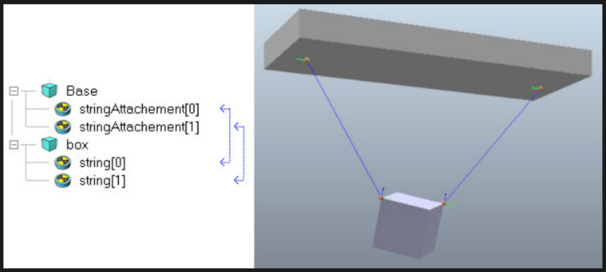
Apart from above 2 main engine differences (algorithms/calculation routines & properties), there are features (or handicaps) only offered by some physics engines. Related to CoppeliaSim, those are described here below:

- **Realistic masses and inertia matrices**: while the MuJoCo, Vortex and Newton engines allow to use realistic values for masses and inertias, the Bullet and ODE engines on the other hand often require *mass and inertia balancing* (i.e. keeping mass and inertia differences between connecting shapes relatively small).
- **Motorized joints**: the *armature* parameter is maybe one of the more important MuJoCo engine specific parameters, and it can have a big impact on how a motorized joint behaves: it adds a rotor inertia (or reflected inertia) to a joint, making the simulation more stable and often increasing physical realism. One should keep in mind that this often requires an increased max. force or torque that the joint can deliver.
- **Consecutive joints**: consecutive joints that are dynamically enabled are only supported with the MuJoCo engine. With the other engines, auxiliary masses should be inserted between consecutive joints.
- **Dynamic content modification**: modifying the content of a scene during simulation, such as adding/removing an object, changing the scene hierarchy, or dynamically resetting an object, is handled seamlessly with all engines, except for the MuJoCo engine that requires a full restart each time: that operation is relatively slow, but will preserve the state of the simulation at that time (e.g. object poses and velocities). One common and frequent situation where this happens are applications of omnidirectional wheel robots, where a dynamic object reset is required in each simulation step.
- **Non-convex shapes**: non-convex meshes are difficult and slow to simulate in general, and often unstable. For that reason they should be avoided at all costs. Some engines however still support them, while others will revert to using their convex representation for simulation (the newton and MuJoCo engines)
- **Static shapes**: static, respondable shapes that are programmatically moved will interact with dynamic, respondable shapes. This interaction includes collision response and friction force transmission. The MuJoCo engine however will not transmit friction forces from static shapes when moved, unless the shape is flagged as kinematic via the kinematic property. In that case, the static shape is represented in MuJoCo as a *mocap* body linked to a dynamic body via a *weld* constraint.
- **Surface velocity**: most engines support body surface velocities, which are often used to simulate a simple conveyor modeled as a cuboid with a surface velocity. This is not possible with the MuJoCo engine, which only supports conveyor belts, where each belt element is static and kinematic, and programmatically moved.
- **Soft bodies**: only the MuJoCo engine supports soft bodies, including clothes and ropes. Those are generated in CoppeliaSim by adding composite items via XML injection. Additional constraints and behaviour can be achieved also via XML injection, knowing that generated XML files have consistent object naming, based on CoppeliaSim object aliases and handles: sim.getObjectAlias(objectHandle,4). Generated MuJoCo files can be found via the mujocoPath property.



[Soft body simulation]

- **Cables/strings and elastics**: only the MuJoCo engine supports tendon-like constraints between two shapes. Those are realized via linked dummies, where the link type is *Dynamics, tendon constraint*. Those tendons can be motorized via a proxy prismatic joint. Other engines will have to revert in modeling a string via two spherical, and one prismatic joint, with appropriate auxiliary masses in-between.



[Strings/elastics]