

SBIRII Delivery Documentation

Agent Rigging

Table of Contents

1	Creating an Agent.....	2
1.1	Your Model.....	3
1.1.1	Export the scene graph.....	3
1.1.2	Physics bodies and shapes.....	3
1.1.3	Physics Joints.....	4
1.1.3.1	Wheels.....	4
1.1.3.2	Suspension.....	5
1.1.3.3	Chassis.....	6
1.2	The agent File.....	6
1.2.1	General Settings.....	6
1.2.2	Drive motors.....	7
1.2.3	Bodies.....	7
1.2.4	Collision sets.....	7
1.2.5	Extern.....	8
1.2.6	Closing tags.....	8
2	Loading the Agent.....	9

Requirements

Before you can create an agent you need to make sure the space is configured to use the physics engine. Please make sure the following blocks of code are in the relevant files.

index.space:

```
<required id="{71CCEAD1-37AD-4572-A267-895207B1082C}" >
  <configuration id="{E0899640-98BA-4d51-A3B5-15C324800CC5}">
    <globalSettings
      scale="1.0"
      driver="ODE"
    />
  </configuration>
</required>
```

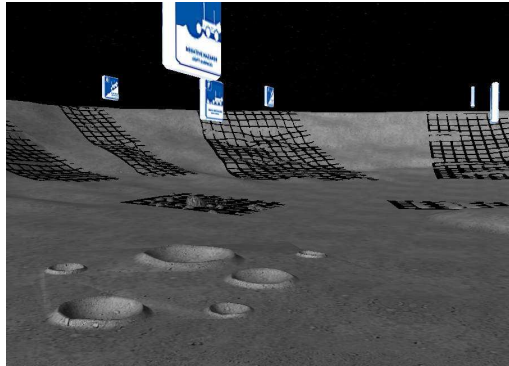
This block enables the physics engine.

sbir2ld.scene:

```
<environment>
  ...
  <gravity name="Moon" x="0" y="-3.92266" z="0" />
  ...
</environment>
```

This Sets gravity to send all physics rigged bodies downwards at 3.9 meters a second.

Last for each ground element we have to give it a physics representation.



The terrain surface

```
<node name="ltp2_softsurface" id="3">
  <position x="-0.721866" y="-1.30912" z="-0.959959" />
  <rotation qx="0" qy="0" qz="0" qw="1" />
  <scale x="1" y="1" z="1" />
  <entity name="ltp2_softsurface" meshFile="ltp2_softsurface.mesh" static="false"
castShadows="false">
    <shape name="ltp2_softsurface" type="mesh" group="0" />
  </entity>
</node>
```

Make sure each of the following have a corresponding <shape> section in the <entity> node.

- ltp2_rocks
- ltp2_hardsurface
- ltp2_softsurface
- ltp2_loosesurface
- ltp2_platform

Creating an Agent.

1. Your Model

NOTE : Because the Agent system makes heavy use of DSS Physics support Agents have to be modeled as separate components rather than 1 single model with sub-meshes. Basically, for each point of movement you wish the agent to have, you need to separate the two pieces that would be either side of the point.

1.1.Export the scene graph

Take all of the Nodes in the .scene file that correspond to your agent and place them in a separate .scene file. eg:

```
<scene>
  <nodes name="ROOT">
    <node name="rlep2_rover2">
      ...
    </node>
  </nodes>
</scene>
```

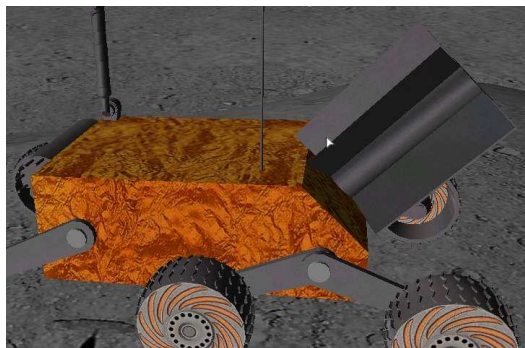
Then place an <extern> tag in the original .scene file to link to your agent. eg

```
<extern src="models/rlep2_opt3/rlep2option3.scene" />
```

1.2.Physics bodies and shapes

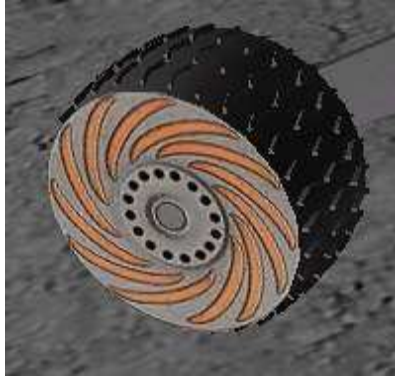
Begin Physics rigging the model, Start relatively simply, change the central body <node> of the agent (chassis on a vehicle) to include the following tags

```
<node name="rlep2rvr2_body" id="68" body="true">
  <position x="-0.000480244" y="0.324706" z="0" />
  <rotation qx="0" qy="0" qz="0" qw="1" />
  <scale x="1" y="1" z="1" />
  <mass value="10" x="1" y="1" z="1" />
  <damping linear="0.200000" angular="0.200000" />
  <entity name="rlep2rvr2_body" id="68" meshFile="rlep2rvr2_body.mesh" static="false" >
    <shape name="rlep2rvr2_body" type="mesh" group="10" />
  </entity>
</node>
```



The chassis

At this point if you load the environment, the node should fall and rest on the ground, showing that it's correctly under the control of the physics engine. Repeat this step for each node that is a part of the wheels and suspension, you should be left with a pile of parts lying on the ground when you start up the space now. For the wheels, use `type="boundingsphere"` instead of `type="mesh"`.



A wheel

1.3. Physics Joints

Now that you have the agent's parts rigged, it's time to start linking them together.

Wheels

The wheels are probably simplest because their center point is the point you wish to use as the position for the axis of the joint. e.g:

```
<joint name="rlpe2opt3_jnt_lfwheel1" type="hinge2" body="rlpe2opt3_lfwheel"
otherBody="rlpe2opt3_lfaxil" >
  <position x="0.699367" y="-0.539326" z="1.10446" />
  <axis2 x="1" y="0" z="0.00" />
  <settings>
    <setting name="suspensionERP">
      <float val="1.0" />
    </setting>
    <setting name="suspensionCFM">
      <float val="0.001" />
    </setting>
  </settings>
  <limit type="angular" low="-0.000001" high="0.000001" axis="0" />
</joint>
```



The wheel axle

Repeat for the following wheels:

- rlep2opt3_lfwheel - rlep2opt3_lfaxil
- rlep2opt3_lmwheel - rlep2opt3_lmaxil
- rlep2opt3_lrwheel - rlep2opt3_lrxil
- rlep2opt3_rfwheel - rlep2opt3_rfaxil
- rlep2opt3_rmwheel - rlep2opt3_rmaxil
- rlep2opt3_rrwheel - rlep2opt3_rrxil

We're using hinge2 type joints so that we can use the suspension variables to simulate some force absorption by the tires. And improve contact with the ground.

The `body=""` and `otherBody=""` attributes specify which two nodes to connect together in this example, the wheels connect to a matching axil.

`<position>` tag specifies the location of the joint in X,Y,Z coordinates.

`<axis>` is used to specify a vector for the joint to act upon, if it's a rotational type joint the axis is the point the joint rotates around, if it's a slider joint the axis is the line the joint follows when sliding apart or together.

`<axis2>` Same as `<axis>` but use on a second axis of a joint. Only used for specific joint types, such as `hinge2` and `universal`.

The `<setting>` tag encompasses a lot of values, please see DSS documentation for a full listing, The two used for `hinge2` joints are.

- `<suspensionERP>` How much per step to correct the joint if it's outside it's constraints.
- `<suspensionCFM>` How far outside the joints constraints the joint is allowed to 'drift'

Combined these two setting allow the system to simulate simple suspension, but visually the movement doesn't match, so this is just used to extremely small movements to try and smooth the rovers progress, i., force that would be absorbed by the wheels. In the example, the movement is effectively zero.

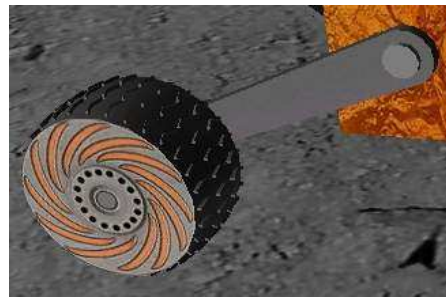
`<limit>` Limits are used to restrict a joints range of movement, in this scene we only use angular limits. When created, a joints position is set to 0, limits are set in relation to this, with low and high limits specifying how far the joint can rotate either + or – from the starting point. Direction of rotation depends on the direction of your specified axis. The axis attribute specifies which axis to apply the limit on on multi axis joint types, in this example, it is placing the limit on axis 0, which would be used for steering in certain vehicle designs but is unneeded in this one.

Repeat for each wheel then move onto securing the axles to the suspension.

Suspension



A suspension arm



A suspension arm

```
<joint name="rlpe2opt3_jnt_rfaxelsusp" type="hinge" body="rlep2opt3_rfaxil"
otherBody="rlep2opt3_rfsusparm" >
  <position x="0.517" y="-0.539326" z="1.10446" />
  <axis x="1" y="0" z="0.00" />
  <limit type="angular" low="-0.00001" high="0.00001" axis="0" />
  <settings>
    <setting name="stopCFM">
      <float val="0.001" />
    </setting>
  </settings>
</joint>
```

For the following axles:

- `rlep2opt3_lfaxil - rlep2opt3_lfsusparm`
- `rlep2opt3_lmaxil - rlep2opt3_lmsusparm`

- rlep2opt3_lraxil - rlep2opt3_lrsusparm
- rlep2opt3_rfaxil - rlep2opt3_rfsusparm
- rlep2opt3_rmaxil - rlep2opt3_rmsusparm
- rlep2opt3_rraxil - rlep2opt3_rrsusparm

The settings for these joints are largely the same only instead of hinge2 we're using single hinge joints. And in the settings we have stopCFM, which has the same purpose as the suspensionCFM value, but is applied to joint limits instead.

NOTE : Determining the position for the joints in this example is actually easy, simply use the position of the first bpd in each pair, the parts where modeled so that their zero point is the connection point for the object.

Chassis

Once these joints have been added the final step is to connect the suspension arms to the Chassis/body of the agent.

```
<joint name="rlpe2opt3_jnt_rfsuspbody" type="hinge" body="rlep2opt3_rfsusparm"
otherBody="rlep2opt3_body" >
  <position x="-0.517" y="-0.252" z="0.47" />
  <axis x="1" y="0" z="0.00" />
  <limit type="angular" low="-0.10001" high="0.10001" axis="0" />
  <settings>
    <setting name="stopCFM">
      <float val="0.001" />
    </setting>
  </settings>
</joint>
```

For the following suspension arms:

- rlep2opt3_lfsusparm - rlep2opt3_body
- rlep2opt3_lmsusparm - rlep2opt3_body
- rlep2opt3_lrsusparm - rlep2opt3_body
- rlep2opt3_rfsusparm - rlep2opt3_body
- rlep2opt3_rmsusparm - rlep2opt3_body
- rlep2opt3_rrsusparm - rlep2opt3_body

You should note the limits on these joints, this is so the rover holds itself up rather than simply belling out, instead. At this point the vehicle should be sitting on it's wheel ready to be driven around, which leads us to the next step, the .agent file.

2. The agent File

2.1. General Settings

At the top of the .agent file is the general settings block, these are the settings most likely to be changed.

```
<general
  steermode="tank"
  centralbody="rlep2opt3_body"
  motorspeed="10" speedunits="radians/second"
  motorforce="15" forceunits="newtonmeters"
  boost="1"
/>
```

the `steermode=""` attribute defines how the agent manager will attempt to control the agent, if it's tank, steering is achieved in a tank like manner, but slowing or reversing the opposite set of wheels to cause to the body to rotate. If set to car, turning is attempted by rotating the specified hinge joints.

`centralbody=""` specifies the central scene node for the agent, it's used to determine orientation and position, generally set to the chassis/body of an agent.

`motorspeed=""` speed of the rover, specified as radians per second. The actual end speed of the vehicles will depend on the diameter of the agent's wheels, (see formula below.)

$$((\pi * 2) / \text{motorspeed}) * \text{wheeldiameter}$$

`motorforce=""` The amount of force the motor will exert to try and propel the rover up to it's full speed, measured in newtonmeters, each motorized joint will be applying this amount of force.

`boost=""` A multiplier used on the motor speed and force, it is advised you do NOT use this value as it is unpredictable on how it will effect agent behavior

2.2. Drive motors.

The joints used to drive the vehicle in tank mode are specified like so.

```
<leftside>
  <wheel>rlpe2opt3_jnt_lfwheel1</wheel>
  <wheel>rlpe2opt3_jnt_lmwheel1</wheel>
  <wheel>rlpe2opt3_jnt_lrwheel1</wheel>
</leftside>
<rightside>
  <wheel>rlpe2opt3_jnt_rfwheel1</wheel>
  <wheel>rlpe2opt3_jnt_rmwheel1</wheel>
  <wheel>rlpe2opt3_jnt_rrwheel1</wheel>
</rightside>
```

`leftside` and `rightside` are used to specify which side of the rover the wheel is on so that it can steer, then `<wheel></wheel>` specifies individual wheels joints.

2.3. Bodies

The bodies listed in this section are used to set what contact properties are used for interacting with the rover.

```
<bodies>
  <body>rlpe2opt3_lfwheel</body>
  <body>rlpe2opt3_lmwheel</body>
  <body>rlpe2opt3_lrwheel</body>
  <body>rlpe2opt3_rfwheel</body>
  <body>rlpe2opt3_rmwheel</body>
  <body>rlpe2opt3_rrwheel</body>
</bodies>
```

2.4. Collision sets

Lets the user set contact properties for collisions, between the bodies specified in `<bodies>` and any other physics body in the scene.

```
<collision>
  <setting name="staticFriction">
    <float val="0.5" />
  </setting>
  <setting name="kineticFriction">
    <float val="0.25" />
  </setting>
  <setting name="restitution">
    <float val="1e-5" />
  </setting>
  <setting name="dustMultiplier">
    <float val="5" />
  </setting>
  <setting name="pyramidFriction">
    <bool val="true" />
  </setting>
</collision>
```

`staticFriction=""` This property is the friction in the direction of contact. This should always be set, as it is the prime property used in simulating the contact. However, this property is

misnamed. In scientific physics simulations, static friction refers to a different concept. The misnaming was due to misunderstanding during early development of the physics simulation.

`kineticFriction=""` This property is the friction in the direction perpendicular to the contact. This allows the physics simulation to model sideways movement differently to forward movement, such as is the case when using tires with tread. If not set, the value for `staticFriction` will be used for sideways motion as well as forward motion.

However, this property is misnamed. In scientific physics simulations, kinetic friction refers to a different concept. The misnaming was due to misunderstanding during early development of the physics simulation.

`restitution=""` This property is the “bounciness” of the contact. At 0, there will be no bounce, while at a value of 1, the object will rebound with the same velocity it collided with. Values larger than 1 will cause the object to rebound at a speed greater than the speed of collision.

`dustMultiplier=""` Indicates how much dust should be generated when colliding with the surface

`pyramidFriction="True"` This is a Boolean property (true or false). It controls the range of values used for the `staticFriction` and `kineticFriction` properties. To use standard scientific values, this property should be set to true.

Additional collision sets can be set after the default one... additional settings in **Bold:**

```
<collision name="hard" >
  <shape>ltp2_rocks</shape>
  <shape>ltp2_hardsurface</shape>
  <shape>ltp2_platform</shape>
  <setting name="staticFriction">
    <float val="0.8" />
  </setting>
  <setting name="kineticFriction">
    <float val="0.25" />
  </setting>
  <setting name="restitution">
    <float val="1e-5" />
  </setting>
  <setting name="dustMultiplier">
    <float val="1" />
  </setting>
  <setting name="pyramidFriction">
    <bool val="true" />
  </setting>
</collision>
```

`name=""` Used as an identifier between collision sets, must be unique between sets.

`<shape>` is used to indicate which surfaces use this set of contact properties.

2.5. Extern

At this point the agent should be fully operational, we can now move the `<extern>` tag from the .scene file and place it in the agent, making sure the model only loads when the agent does.

```
<extern src="models/rlep2_opt3/rlep2option3.scene" />
```

In this case we placed the mesh, scene and texture files in their own folder under models/ so that we can simply copy that folder and the .agent file to use the agent in different spaces.

2.6. Closing tags

Now we just close the tags we opened at the start of the file.

```
</agent>
</chunk>
```

Loading the Agent

The agent file needs to be included in the space file for it to be loaded.

```
<required id="{00519F89-B1DF-43d8-9B01-0EE069ADC16C}" >  
  <configuration id="{00519F89-B1DF-43d8-9B01-0EE069ADC16C}">  
    <agent src="rover2.agent"/>  
    <agent src="rover1.agent"/>  
  </configuration>  
  <required id="{96E29E53-95F9-4bd5-81D6-DB5FFC90675E}" >  
  </required>  
</required>
```

This causes Digital Spaces to load both the Vehicle Agent controller and the Agent Manager. The Vehicle Agent Controller is instructed to create and control agents made from the specified agent files. The Agent Manager provides the implementation of these agents.