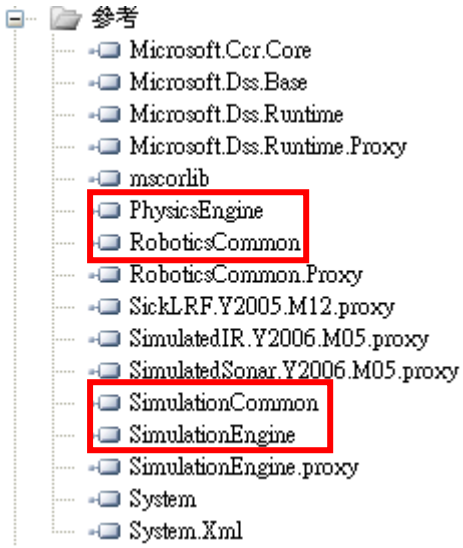
Robot2010 – Development of simulation entity

#Namespace

```
using Microsoft.Robotics.Simulation.Engine;
using Microsoft.Robotics.Simulation;
using Microsoft.Robotics.Simulation.Physics;
using Microsoft.Robotics.PhysicalModel;
```

#Reference



#Insert a VisualEntity

```
VisualEntity entity = new VisualEntity();
entity.State.Assets.Mesh = "table_01.obj";
entity.State.Name = "MyTable";
entity.State.Pose.Position = new Microsoft.Robotics.PhysicalModel.Vector3(0, 2, 0);
SimulationEngine.GlobalInstancePort.Insert(entity);
```

#Insert a child

```
VisualEntity entity = new VisualEntity();
entity.State.Assets.Mesh = "table_01.obj";
entity.State.Name = "MyTable";
entity.State.Pose.Position = new Microsoft.Robotics.PhysicalModel.Vector3(0, 2, 0);

VisualEntity childEntity = new VisualEntity();
childEntity.State.Assets.Mesh = "earth.obj";
childEntity.State.Name = "Child";
childEntity.State.Pose.Position = new Microsoft.Robotics.PhysicalModel.Vector3(1, 1, 0);
```

```
entity.InsertEntity(childEntity);
SimulationEngine.GlobalInstancePort.Insert(entity);
```

#SingleShape Entity

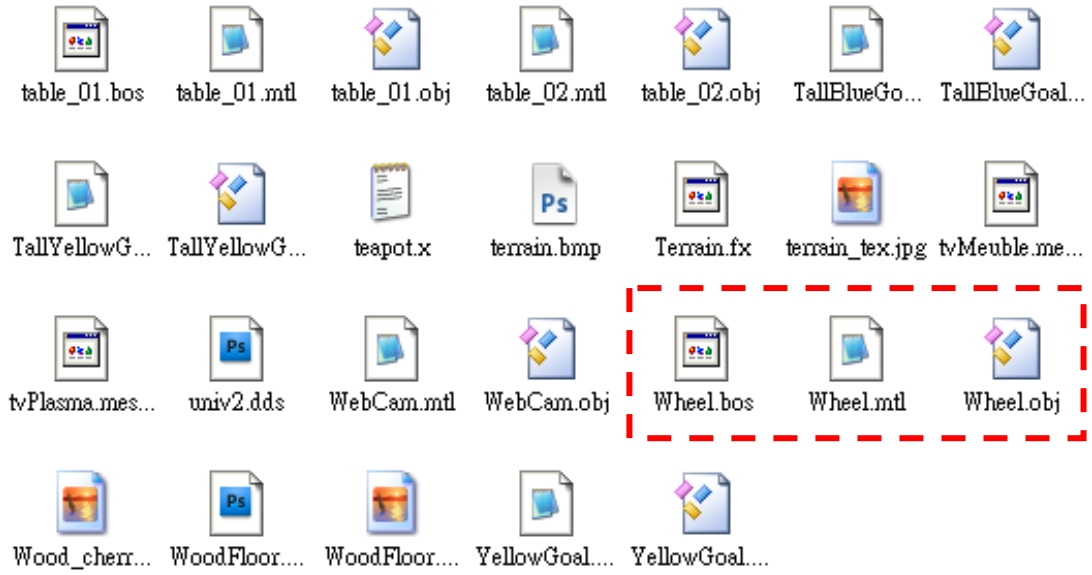
```
BoxShape boxShape = new BoxShape(new BoxShapeProperties(1,new Pose(), new Vector3(1,1,1)));
CapsuleShape capsuleShape = new CapsuleShape(new CapsuleShapeProperties(1, new Pose(), 1, 2));
SphereShape sphereShape = new SphereShape(new SphereShapeProperties(1, new Pose(), 1));

SingleShapeEntity entity = new SingleShapeEntity(boxShape, new Vector3(0, 1, 0));
entity.State.Name = "MySingleShape";
SimulationEngine.GlobalInstancePort.Insert(entity);
```

#MultipleShape Entity

```
BoxShape[] boxes = new BoxShape[3];
boxes[0] =  new BoxShape(new BoxShapeProperties(1,new Pose(), new Vector3(1,1,1)));
boxes[1] =  new BoxShape(new BoxShapeProperties(1,new Pose(new Vector3(0,2,0)), new
Vector3(1,1,1)));
boxes[2] =  new BoxShape(new BoxShapeProperties(1,new Pose(new Vector3(0,4,0)), new
Vector3(1,1,1)));

SphereShape[] spheres = new SphereShape[2];
spheres[0] = new SphereShape(new SphereShapeProperties(1, new Pose(new Vector3(0, 5, 0)), 1));
spheres[1] = new SphereShape(new SphereShapeProperties(1, new Pose(new Vector3(0, 6, 0)), 1));

MultiShapeEntity entity = new MultiShapeEntity(boxes, spheres);
entity.State.Name = "MyMultipleShapeEntity";

SimulationEngine.GlobalInstancePort.Insert(entity);
```

#Custom model

➔ Microsoft Robotics Dev Studio 2008 R2\store\media



#MTL file

newmtl Color_009

Ka 0.000000 0.000000 0.000000

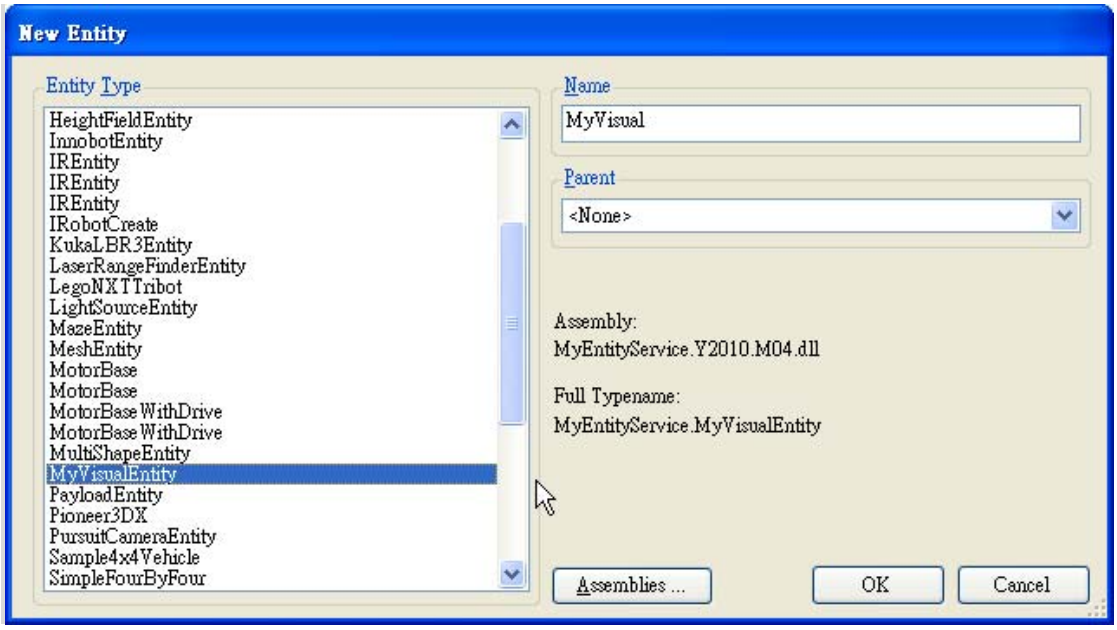
Kd 0.000000 0.000000 0.000000

Ks 0.330000 0.330000 0.330000

map_Kd *Sonor/Metal_Corrogated_Shiny.jpg*

#Custom entity

```
public class MyVisualEntity : VisualEntity
{
    public MyVisualEntity() : base()
    {
        this.State.Assets.Mesh = "table_01.obj";
        this.State.Pose.Position = new Vector3(0, 3f, 0);
    }
}
```



#MyTable entity

```
public class MyTableEntity : MultiShapeEntity
{
        public MyTableEntity() { }

        /// <summary>
        /// Custom constructor, programmatically builds physics primitive shapes to describe
        /// a particular table.
        /// </summary>
        /// <param name="position"></param>
        public MyTableEntity(Vector3 position)
        {
            State.Pose.Position = position;
            State.Assets.Mesh = "table_01.obj";
            float tableHeight = 0.65f;
            float tableWidth = 1.05f;
            float tableDepth = 0.7f;
            float tableThinkness = 0.03f;
            float legThickness = 0.03f;
            float legOffset = 0.05f;

            // add a shape for the table surface
            BoxShape tableTop = new BoxShape(
                new BoxShapeProperties(30,
```

```
                new Pose(new Vector3(0, tableHeight, 0)),
                new Vector3(tableWidth, tableThinkness, tableDepth))
            );

            // add a shape for the left leg
            BoxShape tableLeftLeg = new BoxShape(
                new BoxShapeProperties(10, // mass in kg
                new Pose(
                    new Vector3(-tableWidth / 2 + legOffset, tableHeight / 2, 0)),
                new Vector3(legThickness, tableHeight + tableThinkness, tableDepth))
            );

            BoxShape tableRightLeg = new BoxShape(
                new BoxShapeProperties(10, // mass in kg
                new Pose(
                    new Vector3(tableWidth / 2 - legOffset, tableHeight / 2, 0)),
                new Vector3(legThickness, tableHeight + tableThinkness, tableDepth))
            );

            BoxShapes = new List<BoxShape>();
            BoxShapes.Add(tableTop);
            BoxShapes.Add(tableLeftLeg);
            BoxShapes.Add(tableRightLeg);
        }
}
```

#DifferentialDriveEntity entity

```
public class MyDifferentialDriveEntity : DifferentialDriveEntity
    {
        public MyDifferentialDriveEntity () { }
        public MyDifferentialDriveEntity (Vector3 initialPos)
        {
            MASS = 9;
            // the default settings are approximating a Pioneer 3-DX activMedia robot
            CHASSIS_DIMENSIONS = new Vector3(0.393f, 0.18f, 0.40f);
            CHASSIS_CLEARANCE = 0.05f;
            FRONT_WHEEL_RADIUS = 0.08f;
            CASTER_WHEEL_RADIUS = 0.025f; // = CHASSIS_CLEARANCE / 2; // to keep things simple
we make caster a bit bigger
```

```
            FRONT_WHEEL_WIDTH = 4.74f;  //not used
            CASTER_WHEEL_WIDTH = 0.02f; //not used
            FRONT_AXLE_DEPTH_OFFSET = -0.05f; // distance of the axle from the center of robot

            base.State.Name = "MotorBaseWithThreeWheels";
            base.State.MassDensity.Mass = MASS;
            base.State.Pose.Position = initialPos;
            base.State.Assets.Mesh = "Pioneer3dx.bos";
            base.WheelMesh = "PioneerWheel.bos";

            // reference point for all shapes is the projection of
            // the center of mass onto the ground plane
            // (basically the spot under the center of mass, at Y = 0, or ground level)

            // chassis position
            BoxShapeProperties motorBaseDesc = new BoxShapeProperties("chassis", MASS,
                new Pose(new Vector3(
                0, // Chassis center is also the robot center, so use zero for the X axis offset
                CHASSIS_CLEARANCE + CHASSIS_DIMENSIONS.Y / 2, // chassis is off the ground and its center is DIM.Y/2 above the clearance
                0)), // no offset in the z/depth axis, since again, its center is the robot center
                CHASSIS_DIMENSIONS);

            motorBaseDesc.Material = new MaterialProperties("high friction", 0.0f, 1.0f, 20.0f);
            motorBaseDesc.Name = "Chassis";
            ChassisShape = new BoxShape(motorBaseDesc);

            // rear wheel is also called the caster
            CASTER_WHEEL_POSITION = new Vector3(0, // center of chassis
                CASTER_WHEEL_RADIUS, // distance from ground
                CHASSIS_DIMENSIONS.Z / 2 - CASTER_WHEEL_RADIUS); // all the way at the back of the robot

            // NOTE: right/left is from the perspective of the robot, looking forward

            FRONT_WHEEL_MASS = 0.10f;
```

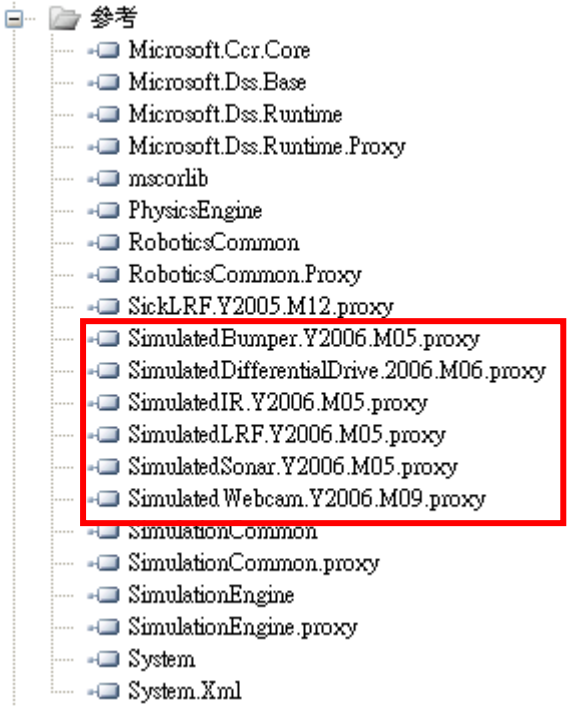
```
            RIGHT_FRONT_WHEEL_POSITION = new Vector3(
                CHASSIS_DIMENSIONS.X / 2 + 0.01f - 0.05f,// left of center
                FRONT_WHEEL_RADIUS,// distance from ground of axle
                FRONT_AXLE_DEPTH_OFFSET); // distance from center, on the z-axis

            LEFT_FRONT_WHEEL_POSITION = new Vector3(
                -CHASSIS_DIMENSIONS.X / 2 - 0.01f + 0.05f,// right of center
                FRONT_WHEEL_RADIUS,// distance from ground of axle
                FRONT_AXLE_DEPTH_OFFSET); // distance from center, on the z-axis

            MotorTorqueScaling = 20;
        }
    }
```

#Entity with sensors

```
    public class MyDifferentialEntityWithSensor : MyDifferentialDriveEntity
    {
        LaserRangeFinderEntity _laser;
        public LaserRangeFinderEntity Laser
        {
            get { return _laser; }
            set { _laser = value; }
        }
        BumperArrayEntity _bumperArray;
        public BumperArrayEntity BumperArray
        {
            get { return _bumperArray; }
            set { _bumperArray = value; }
        }
        CameraEntity _cam;
        public CameraEntity Cam
        {
            get { return _cam; }
            set { _cam = value; }
        }
        public MyDifferentialEntityWithSensor(){}
        public MyDifferentialEntityWithSensor(Vector3 initialPos) : base(initialPos)
        {
```

```
            //Laser range finder
            _laser = new LaserRangeFinderEntity(
                new Pose(new Vector3(0, 0.30f, 0)));
            _laser.State.Name = this.State.Name + "_lrf";
            _laser.LaserBox.State.DiffuseColor = new Vector4(0.25f, 0.25f, 0.8f, 1.0f);
            //Bumpers
            BoxShape frontBumper = new BoxShape(
                new BoxShapeProperties("front",
                    0.001f,
                    new Pose(new Vector3(0, 0.05f, -0.25f)),
                    new Vector3(0.40f, 0.03f, 0.03f)
                )
            );
            frontBumper.State.DiffuseColor = new Vector4(0.1f, 0.1f, 0.1f, 1.0f);
            BoxShape rearBumper = new BoxShape(
                new BoxShapeProperties("rear",
                    0.001f,
                    new Pose(new Vector3(0, 0.05f, 0.25f)),
                    new Vector3(0.40f, 0.03f, 0.03f)
                )
            );
            rearBumper.State.DiffuseColor = new Vector4(0.1f, 0.1f, 0.1f, 1.0f);
            frontBumper.State.EnableContactNotifications = true;
            rearBumper.State.EnableContactNotifications = true;
            _bumperArray = new BumperArrayEntity(frontBumper, rearBumper);
            _bumperArray.State.Name = this.State.Name + "_bumper";
            //Camera
            _cam = new CameraEntity(320, 240, CameraEntity.CameraModelType.AttachedChild);
            _cam.State.Name = this.State.Name + "_came";
            _cam.State.Pose.Position = new Vector3(0.0f, 0.5f, 0.0f);
            _cam.IsRealTimeCamera = true;

            this.InsertEntity(_laser);
            this.InsertEntity(_bumperArray);
            this.InsertEntity(_cam);
        }
    }
```

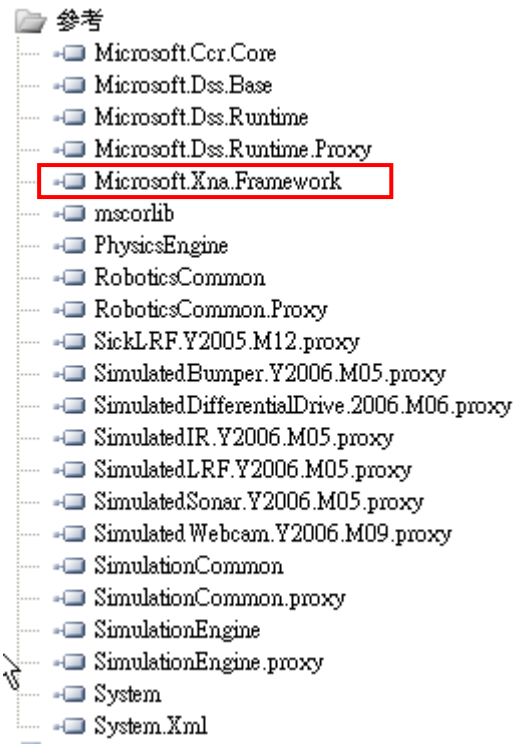
#Sensors with services




```
using drive = Microsoft.Robotics.Services.Simulation.Drive.Proxy;
using lrf = Microsoft.Robotics.Services.Simulation.Sensors.LaserRangeFinder.Proxy;
using bumper = Microsoft.Robotics.Services.Simulation.Sensors.Bumper.Proxy;
using simwebcam = Microsoft.Robotics.Services.Simulation.Sensors.SimulatedWebcam.Proxy;

MyDifferentialEntityWithSensor entity = new MyDifferentialEntityWithSensor(new Vector3(5, 2,
5));
drive.Contract.CreateService(ConstructorPort, "http://localhost/" + entity.State.Name,
                Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
                    "http://localhost/" + entity.State.Name));
lrf.Contract.CreateService(ConstructorPort, "http://localhost/" + entity.Laser.State.Name,
               Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
               "http://localhost/" + entity.Laser.State.Name));
bumper.Contract.CreateService(ConstructorPort, "http://localhost/" +
                entity.BumperArray.State.Name,
                Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
                "http://localhost/" + entity.BumperArray.State.Name));
simwebcam.Contract.CreateService(ConstructorPort, "http://localhost/" +
                entity.Cam.State.Name,
                Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
                    "http://localhost/" + entity.Cam.State.Name)
            );
```

```
SimulationEngine.GlobalInstancePort.Insert(entity);
```

#Entity with joint




```
public override void Initialize(Microsoft.Xna.Framework.Graphics.GraphicsDevice device,
PhysicsEngine physicsEngine)
        {
            base.Initialize(device, physicsEngine);

            if ((_cam.ParentJoint != null) && (_cam.ParentJoint.InternalHandle != (IntPtr)0))
                PhysicsEngine.DeleteJoint((PhysicsJoint)_cam.ParentJoint);

            JointAngularProperties angular = new JointAngularProperties();
            angular.TwistMode = JointDOFMode.Limited;
            angular.Swing1Mode = JointDOFMode.Locked;
            angular.Swing2Mode = JointDOFMode.Locked;
            angular.LowerTwistLimit = new JointLimitProperties(MathTool.DegreesToRadians(-90),
500000, new SpringProperties(500000, 100000, 0));
            angular.UpperTwistLimit = new JointLimitProperties(MathTool.DegreesToRadians(90),
500000, new SpringProperties(500000, 100000, 0));
            angular.TwistDrive = new JointDriveProperties(
                JointDriveMode.Position,
```

```
                new SpringProperties(500000, 100000, 0),
                1000000);

            EntityJointConnector[] connectors = new EntityJointConnector[2]
                {
                    new EntityJointConnector(null, new Vector3(1,0,0), new Vector3(0,1,0), new
Vector3(0, 0,0)),
                    new EntityJointConnector(null, new Vector3(1,0,0), new Vector3(0,1,0), new
Vector3(0,0,0))
                };
            PhysicsJoint jointInstance = null;
            jointInstance = PhysicsJoint.Create(new JointProperties(angular, connectors));
            jointInstance.State.Name = this.State.Name + "_Joint";
            jointInstance.State.EnableCollisions = false;
            jointInstance.State.Connectors[0].Entity = this;
            jointInstance.State.Connectors[1].Entity = _cam;

            _cam.ParentJoint = jointInstance;
            PhysicsEngine.InsertJoint((PhysicsJoint)_cam.ParentJoint);
            ((PhysicsJoint)this.Cam.ParentJoint).SetAngularDriveOrientation(MathTool.CreateQ
            uaternionFromRPYAngles(45, 0, 0));
}

public void SetAngle(float degree)
{
((PhysicsJoint)this.Cam.ParentJoint).SetAngularDriveOrientation(MathTool.CreateQuaternionFro
mRPYAngles(degree, 0, 0));
}

static public class MathTool
    {
        static public Quaternion CreateQuaternionFromRPYAngles(float x, float y, float z)
        {
            Quaternion q = new Quaternion();
            double roll = DegreesToRadians(x);
            double pitch = DegreesToRadians(y);
            double yaw = DegreesToRadians(z);
```

```
            double cyaw, cpitch, croll, syaw, spitch, sroll;
            double cyawcpitch, syawspitch, cyawspitch, syawcpitch;

            cyaw = Math.Cos(0.5f * yaw);
            cpitch = Math.Cos(0.5f * pitch);
            croll = Math.Cos(0.5f * roll);
            syaw = Math.Sin(0.5f * yaw);

            spitch = Math.Sin(0.5f * pitch);
            sroll = Math.Sin(0.5f * roll);

            cyawcpitch = cyaw * cpitch;
            syawspitch = syaw * spitch;
            cyawspitch = cyaw * spitch;
            syawcpitch = syaw * cpitch;

            q.W = (float)(cyawcpitch * croll + syawspitch * sroll);
            q.X = (float)(cyawcpitch * sroll + syawspitch * croll);
            q.Y = (float)(cyawspitch * croll + syawcpitch * sroll);
            q.Z = (float)(syawcpitch * croll + cyawspitch * sroll);

            return q;
        }

        static public float DegreesToRadians(float deg)
        {
            return (float)(deg * Math.PI / 180.0f);
        }

        static public float RadiansToDegrees(float rad)
        {
            return (float)(rad * 180.0f / Math.PI);
        }

    }
```