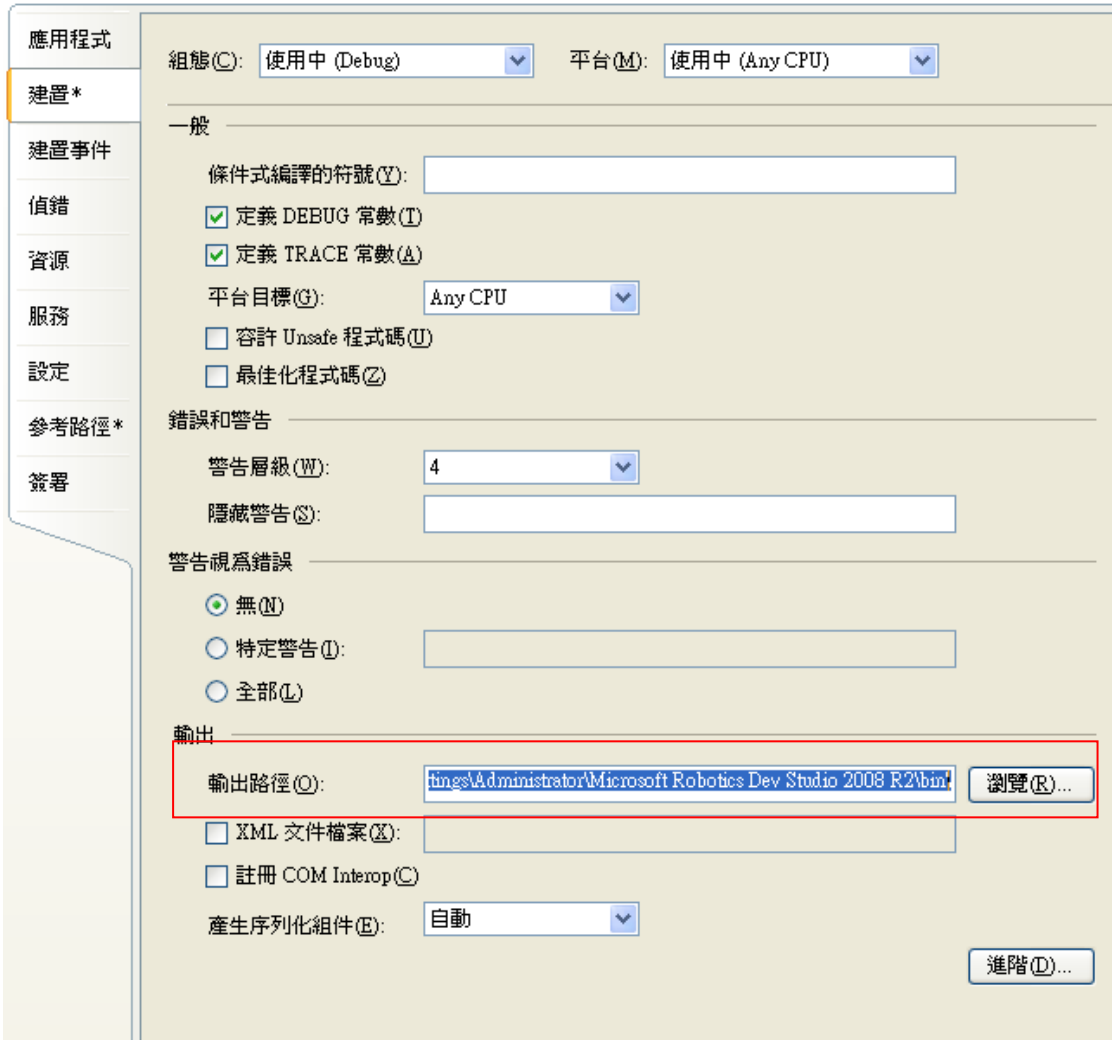
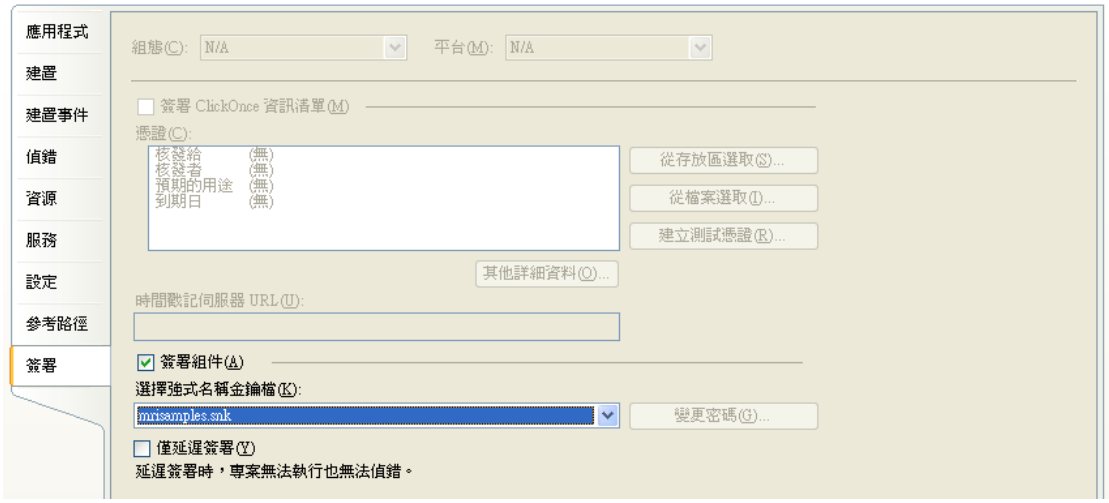
Robot 2010 SimulatedEntity Service 2010/04/26

#My entity library

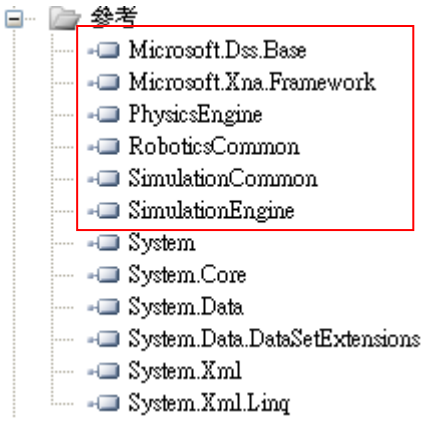
Set output path



Add key



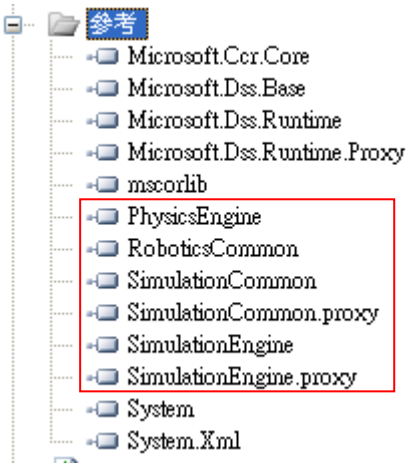
Add basic references



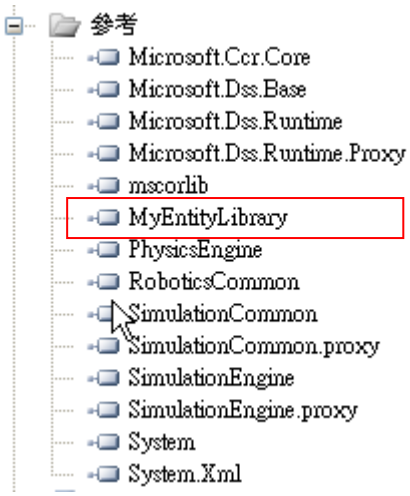

```
using Microsoft.Robotics.Simulation.Engine;
using Microsoft.Robotics.Simulation;
using Microsoft.Robotics.Simulation.Physics;
using Microsoft.Robotics.PhysicalModel;
using xna = Microsoft.Xna.Framework;
```

#Service for simulation entity

Add basic references



Add reference for your simulation entity



Using namespace

```
using simengine = Microsoft.Robotics.Simulation.Engine;
using myentity = MyEntityLibrary;
using Microsoft.Dss.Core.DsspHttp;
using dssp = Microsoft.Dss.ServiceModel.Dssp;
```

Declare variables

```
myentity.MyDifferentialEntityWithSensor _entity;
simengine.SimulationEnginePort _notificationTarget;
```

Initial and subscribe

```
_notificationTarget = new simengine.SimulationEnginePort();
simengine.SimulationEngine.GlobalInstancePort.Subscribe(ServiceInfo.PartnerList,
_notificationTarget);
```

Add listener

```
Activate(new Interleave(
                new TeardownReceiverGroup
                (
Arbiter.Receive<simengine.InsertSimulationEntity>(false, _notificationTarget,
InsertEntityNotificationHandlerFirstTime),
Arbiter.Receive<dssp.DsspDefaultDrop>(false, _mainPort, DefaultDropHandler)
                ),
                new ExclusiveReceiverGroup(),
                new ConcurrentReceiverGroup()
));
```

InsertEntityNotificationHandlerFirstTime

```
void InsertEntityNotificationHandlerFirstTime(simengine.InsertSimulationEntity ins)
        {
            InsertEntityNotificationHandler(ins);
            base.Start();
            MainPortInterleave.CombineWith(
                new Interleave(
                    new TeardownReceiverGroup(),
                    new ExclusiveReceiverGroup(
                        Arbiter.Receive<simengine.InsertSimulationEntity>(true,
_notificationTarget, InsertEntityNotificationHandler),
                        Arbiter.Receive<simengine.DeleteSimulationEntity>(true,
_notificationTarget, DeleteEntityNotificationHandler)
                    ),
                    new ConcurrentReceiverGroup()
                )
            );
        }
```

InsertEntityNotificationHandler & DeleteEntityNotificationHandler

```
void InsertEntityNotificationHandler(simengine.InsertSimulationEntity ins)
{
            _entity = (myentity.MyDifferentialEntityWithSensor)ins.Body;
            _entity.ServiceContract = Contract.Identifier;
}

void DeleteEntityNotificationHandler(simengine.DeleteSimulationEntity del)
{
            _entity = null;
}
```

Service communacation

```
    [DataContract]
    public class SetSickJointAngleRequest
    {
        float _ degree;
        [DataMember]
```

```
        public float Degree
        {
            get { return _degree; }
            set { _ degree = value; }
        }

        public SetSickJointAngleRequest() { }
        public SetSickJointAngleRequest(float degree)
        {
            _ degree = degree;
        }
    }

public class SetSickJointAngleMessage : Upsert<SetSickJointAngleRequest,
PortSet<DefaultUpsertResponseType, Fault>>
    {
        public SetSickJointAngleMessage() : base() { }

        public SetSickJointAngleMessage(SetSickJointAngleRequest body)
            : base(body)
        {
        }

        public SetSickJointAngleMessage(SetSickJointAngleRequest body,
PortSet<DefaultUpsertResponseType, Fault> responsePort)
            : base(body, responsePort)
        {
        }
    }
```

Add the message type to service port

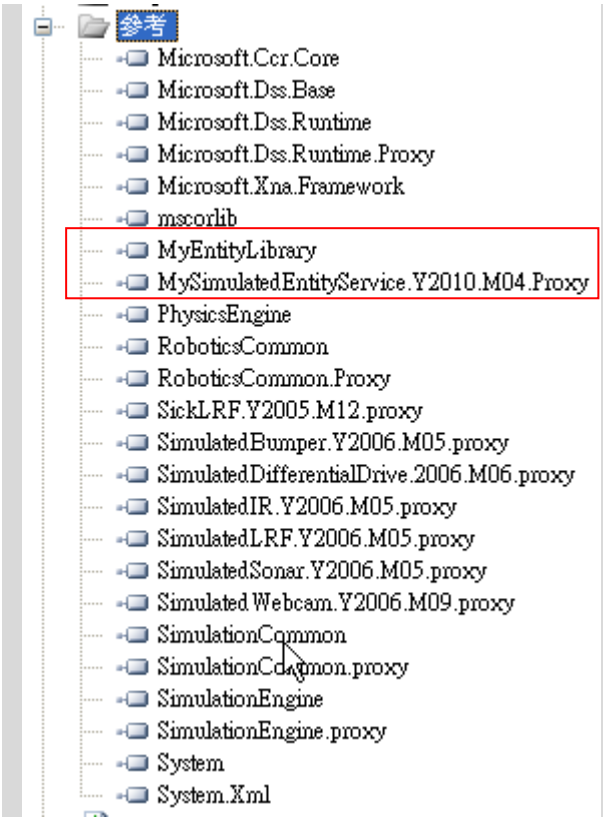
```
    [ServicePort]
    public class MySimulatedEntityServiceOperations
        : PortSet<DsspDefaultLookup,
        DsspDefaultDrop,
        Get,
        Subscribe,
        SetSickJointAngleMessage>
    {
```

```
    }
```

Add handler

```
        [ServiceHandler]
        public void SetSickAngleHandler(SetSickJointAngleMessage setAngle)
        {
            _entity.SetAngle(setAngle.Body.Degree);
            setAngle.ResponsePort.Post(new DefaultUpsertResponseType());
        }
```

#Add service



Insert entity

```
MyEntityLibrary.MyDifferentialEntityWithSensors entity = new
MyEntityLibrary.MyDifferentialEntityWithSensors(new Vector3(20, 1, 20));
MySimulatedEntityService.Proxy.Contract.CreateService(
                ConstructorPort, "http://localhost/" + entity.State.Name,
               Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
                "http://localhost/" + entity.State.Name));
SimulationEngine.GlobalInstancePort.Insert(entity);
```

#Entity control service

Inset entity message

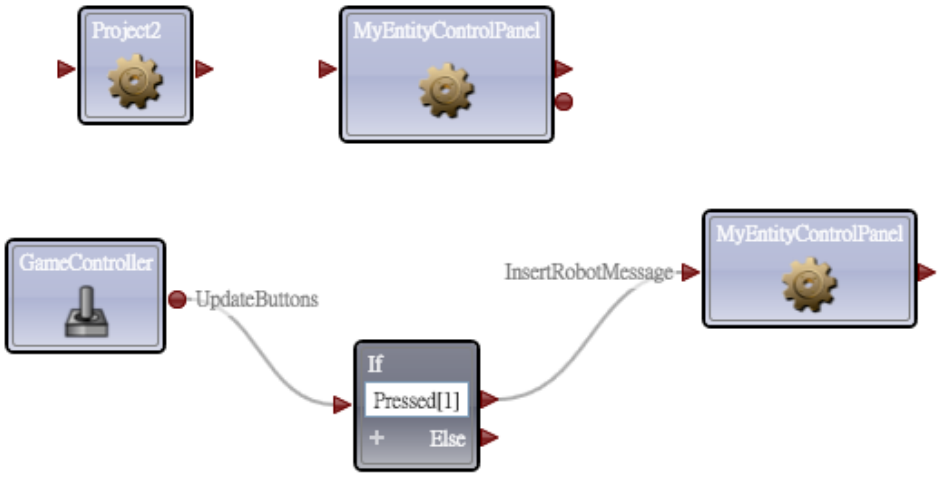
```
    [DataContract]
    public class InsertRobotRequest
    {
        float _x;
        [DataMember]
        public float X
        {
            get { return _x; }
            set { _x = value; }
        }
        float _y;
        [DataMember]
        public float Y
        {
            get { return _y; }
            set { _y = value; }
        }
        float _z;
        [DataMember]
        public float Z
        {
            get { return _z; }
            set { _z = value; }
        }
        public InsertRobotRequest() { }
    }
    public class InsertRobotMessage : Upsert<InsertRobotRequest,
PortSet<DefaultUpsertResponseType, Fault>>
    {
        public InsertRobotMessage() { }
    }
```

Add handler

```
        [ServiceHandler]
        public void InsetEntityHandler(InsertRobotMessage ins)
        {
            _entity = new MyEntityLibrary.MyDifferentialEntityWithSensors(
                new Vector3(ins.Body.X,ins.Body.Y,ins.Body.Z));
```

```
            SimulationEngine.GlobalInstancePort.Insert(_entity);
        }
```

Test



Set degree handler

```
    [DataContract]
    public class SetSickDegreeRequest
    {
        float _degree;
        [DataMember]
        public float Degree
        {
            get { return _degree; }
            set { _degree = value; }
        }
        public SetSickDegreeRequest() { }
    }
    public class SetSickDegreeMessage : Upsert<SetSickDegreeRequest,
PortSet<DefaultUpsertResponseType, Fault>>
    {
        public SetSickDegreeMessage() { }
    }
```

Add handler

```
        [ServiceHandler]
        public void SetSickDegreeHandler(SetSickDegreeMessage set)
```

```
        {
            _entity.SetAngle(set.Body.Degree);
        }
```

Test