

## Robot 2010 SimulatedEntity Service 2010/04/26

#My entity library

Set output path

The screenshot shows the 'Build' configuration page in Visual Studio. The 'Output' section is highlighted with a red box. The 'Output path' field is set to 't:\Program Files\Microsoft Robotics Dev Studio 2008 R2\bin\'. The 'Generate XML file' and 'Register COM Interop' checkboxes are unchecked. The 'Generate serialized components' dropdown is set to 'Automatic'. The 'Advanced' button is visible at the bottom right.

應用程式  
建置\*  
建置事件  
偵錯  
資源  
服務  
設定  
參考路徑\*  
簽署

組態(C): 使用中 (Debug) 平台(M): 使用中 (Any CPU)

一般

條件式編譯的符號(Y):

定義 DEBUG 常數(D)  
 定義 TRACE 常數(A)

平台目標(G): Any CPU

容許 Unsafe 程式碼(U)  
 最佳化程式碼(O)

錯誤和警告

警告層級(W): 4

隱藏警告(S):

警告視為錯誤

無(N)  
 特定警告(I):  
 全部(L)

輸出

輸出路徑(O): t:\Program Files\Microsoft Robotics Dev Studio 2008 R2\bin\ 瀏覽(B)...

XML 文件檔案(X):  
 註冊 COM Interop(C)

產生序列化組件(E): 自動

進階(A)...

Add key

The screenshot shows the 'Signing' configuration page in Visual Studio. The 'Signing' section is active. The 'Sign assembly' checkbox is checked. The 'Strong name key' dropdown is set to 'mismamples.snk'. The 'Delay signing' checkbox is unchecked. The 'Time stamping URL' field is empty. The 'Other detailed information' button is visible.

應用程式  
建置  
建置事件  
偵錯  
資源  
服務  
設定  
參考路徑  
簽署

組態(C): N/A 平台(M): N/A

簽署 ClickOnce 資訊清單(M)

憑證(C):

核發給	(無)
核發者	(無)
預期的用途	(無)
到期日	(無)

從存放區選取(S)...  
從檔案選取(I)...  
建立測試憑證(R)...

其他詳細資料(O)...

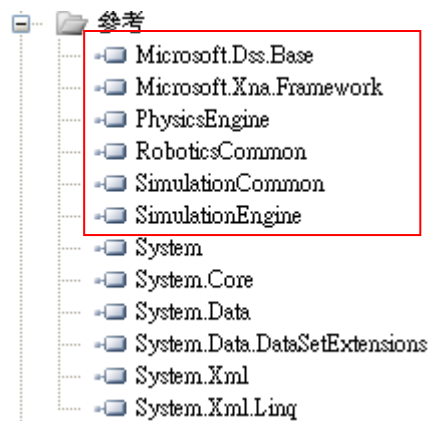
時間戳記伺服器 URL(U):

簽署組件(A)

選擇強式名稱金鑰檔(K): mismamples.snk 變更密碼(G)...

僅延遲簽署(Y)  
延遲簽署時，專案無法執行也無法偵錯。

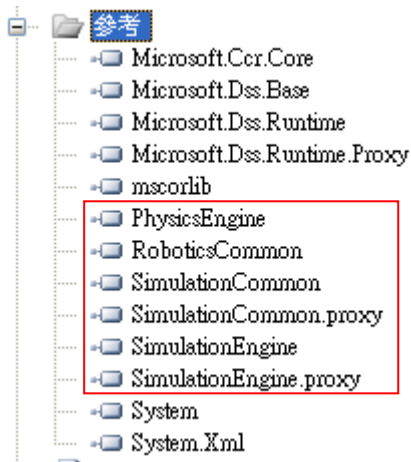
## Add basic references



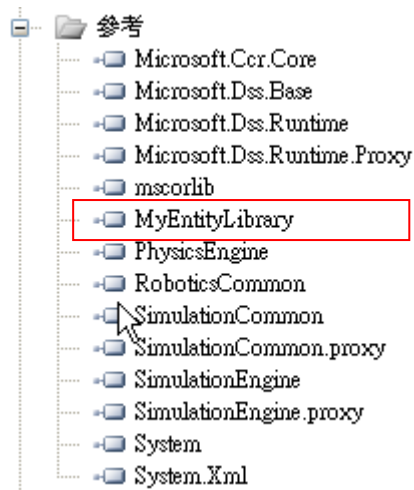
```
using Microsoft.Robotics.Simulation.Engine;  
using Microsoft.Robotics.Simulation;  
using Microsoft.Robotics.Simulation.Physics;  
using Microsoft.Robotics.PhysicalModel;  
using xna = Microsoft.Xna.Framework;
```

## #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 communication

```
[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)
```

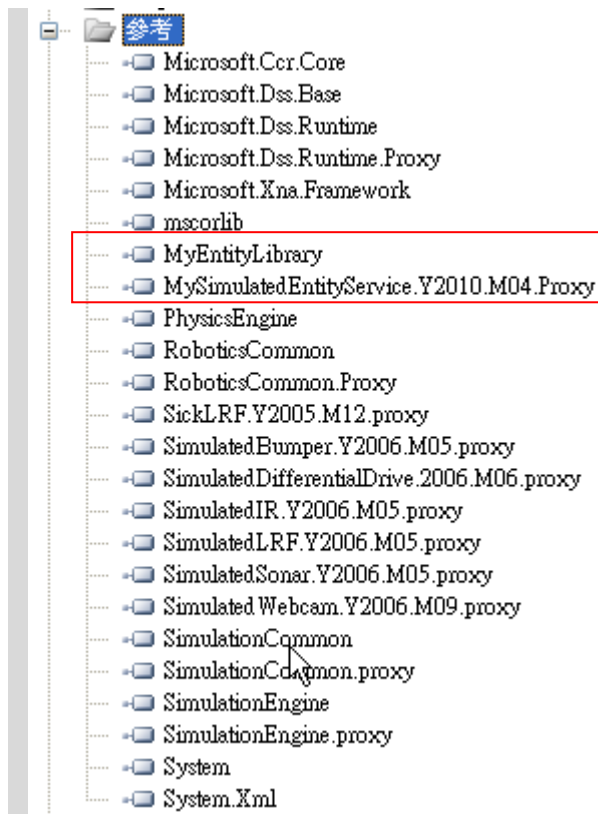
```
{
```

```
    _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() { }
}
```

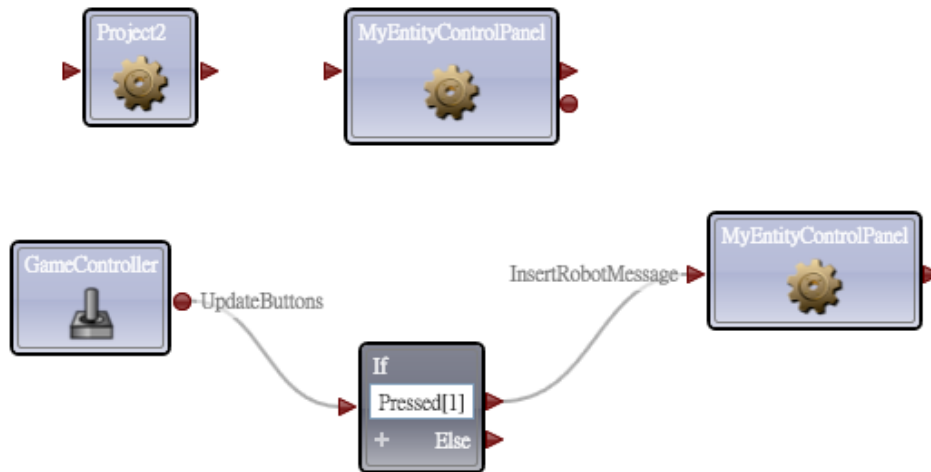
## Add handler

```
[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

