



Getting Started

A basic tutorial on setting up a simulation.

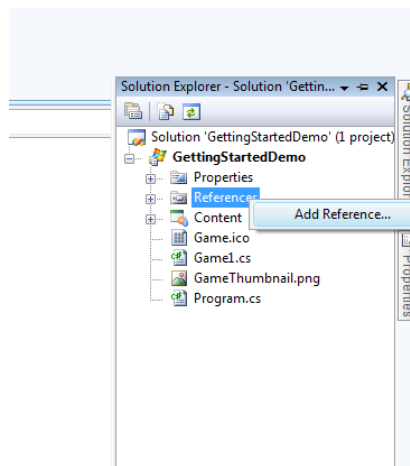
1 | Setting up the Project

The following assumes that you have installed Visual C# 2008 Express and XNA Game Studio 3.1. The GettingStartedDemo shows the completed result of this tutorial and is available on the website with source. The BasicSetupDemo is the result of only the first parts of this tutorial, showing a project and a few entities.

To begin, create a new XNA 3.1 Windows Game project in Visual C# Express. This creates a simple game template into which BEPUpHysics will be integrated. Running the project now will show the default cornflower blue background.

Go to the BEPUpHysics website (www.bepuphysics.com) and download the latest XNA 3.1 Windows version of the library. The two files you need for your project from the download are the BEPUpHysics.dll and BEPUpHysics.xml. The .dll file is the library itself and the .xml file is its associated documentation which is read by Visual Studio's intellisense.

Put the two files together somewhere convenient in your project that can be gotten to easily. Re-open Visual Studio and add the BEPUpHysics.dll to the project as a reference. This is done by going to the solution explorer, locating the references, right clicking, and choosing "Add Reference." Find the BEPUpHysics.dll from the Browse tab and click OK.



The remainder of the tutorial will assume that you are working on a Windows project, but you can create an Xbox 360 copy of the project by right clicking on the project name and clicking on the Create Copy Of Project for Xbox 360. You will also need to get the BEPUpHysics library for the Xbox 360, available on the BEPUpHysics website. The Xbox 360 library can be added the Xbox 360 project similarly to the above example.

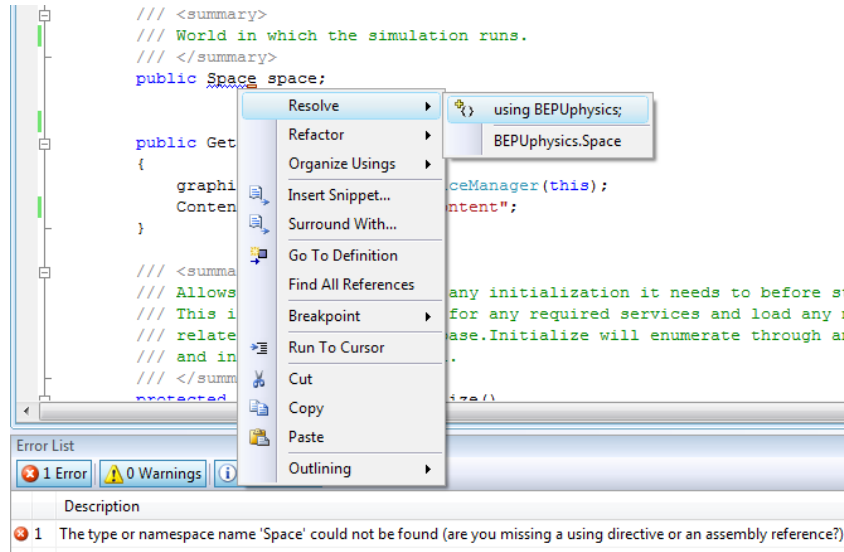
2 | Setting Up a Simulation

Before a simulation can start, a place for the simulation to occur must be created. This is handled by the Space class of BEPUphysics. Other simulation objects can be added to and removed from a Space instance. Creating a space is very simple; first, set up a field in your game for the space, and then construct it in the game's LoadContent method:

```
space = new Space();
```

Now your simulation objects have a place to live.

If you get an error saying "The type or namespace name 'Space' could not be found," add BEPUphysics to the set of using statements at the top of the file. This can be done automatically by right clicking on the error-causing Space reference, clicking Resolve, then clicking the "using BEPUphysics;" option. If the Resolve option is not available, ensure that you have added the BEPUphysics.dll to your project's references.



BEPUphysics needs to be told that time is passing to allow objects to move. In your game's Update method, add in the following line:

```
space.Update(gameTime);
```

When you run the game at this point, BEPUphysics is invisibly working in the background, updating the world every frame.

2.A | Adding Entities

The world is pretty boring without anything in it. The Entity class represents common physical objects populating the space. You can create a variety of different Entity types, including Box, Cylinder, Sphere, Capsules, and others which can be found in the `BEPUPhysics.Entities` namespace. If during the following process you encounter a 'type not found' error similar to the above with Space, you can right click and use the Resolve functionality as before.

All entities can either be dynamic or kinematic. Dynamic entities fall, get knocked around, bounce, and slide as expected. Kinematic entities are like dynamic entities, but can be thought of as having infinite mass. They will not change their velocities as the result of any collision or interaction unless explicitly told to do so. A kinematic entity moving towards a dynamic entity will simply push the dynamic entity around. If a kinematic entity encounters another kinematic entity, they pass through each other.

The only difference between constructing a kinematic entity and a dynamic entity is the last parameter of the constructor. Dynamic constructors have a "mass" parameter, while kinematic constructors do not. You can change between dynamic and kinematic later by calling the entity's `BecomeKinematic` and `BecomeDynamic` methods.

Kinematic entities are a good choice for static and structural shapes, like the ground. You can make a kinematic box representing the ground by putting the following in the game's `LoadContent` method:

```
Box ground = new Box(Vector3.Zero, 30, 1, 30);
```

The first parameter represents the position of the box and the following three parameters are its width, height, and length. Now add the ground to the space:

```
space.Add(ground);
```

Throw some extra dynamic cubes at the ground too:

```
space.Add(new Box(new Vector3(0, 4, 0), 1, 1, 1, 1));  
space.Add(new Box(new Vector3(0, 8, 0), 1, 1, 1, 1));  
space.Add(new Box(new Vector3(0, 12, 0), 1, 1, 1, 1));
```

The last parameter specified is the mass as explained earlier, making these entities dynamic.

The simulation's gravity acceleration vector defaults to (0, 0, 0), so to make things move, set the gravity to a more earth-like value by going into the space's simulation settings:

```
space.SimulationSettings.MotionUpdate.Gravity = new Vector3(0, -9.81f, 0);
```

You can adjust a wide range of behavior in the engine within the space's simulation settings. Take a moment to look around in the various categories to familiarize yourself with the layout.

3 | Basic Interaction with BEPUphysics

If you run the game now, BEPUphysics is running but nothing is visible. To remedy this, attach some graphics to the entities. A basic implementation of a `DrawableGameComponent` that follows an Entity and a Camera which manages the viewpoint of the user are available in the `GettingStartedDemo`. This document will not go in depth on how to set up these systems, but will describe how these pieces interact with the physics engine.

3.A | Getting Entity Position for Rendering

The `Draw` method of the `EntityModel` class has the following line of code at the beginning:

```
Matrix worldMatrix = Transform * entity.worldTransform;
```

This line defines what transformation to use for rendering the model. The transform variable is just a local space transformation that can be used to adjust the model in case it needs to be scaled, moved, or rotated. A common problem is a model being loaded in that wasn't centered on the origin in the modeling application. This extra transform would allow you to re-center the model without going back into the application.

The line of code grabs the entity's current world transformation. This matrix represents a rigid transformation; that is, it only includes orientation and translation. It is a convenient representation of the entity's state for rendering, though you can also directly access the entity's `CenterPosition`, `OrientationMatrix`, and a variety of other properties. Look around in the entity's property list to get an idea of what is available.

3.B | Firing a Box

In addition to the basic camera manipulation controls in the `Camera` class, there is a section in the game's `Update` method which creates a box when the left mouse button is clicked. This looks like the box creation code shown previously:

```
Box toAdd = new Box(camera.position, 1, 1, 1, 1);
```

However, the velocity of the box is set as well. This is done by accessing another one of the entity's properties—the `LinearVelocity`:

```
toAdd.LinearVelocity = camera.worldMatrix.Forward * 10;
```

The box will fly off in the direction that the camera is facing. Try changing the speed and changing other properties to see the result.

4 | Adding an Environment

In the same way that entities can be added to the space, a variety of other types can be added as well. One common type is known as the `StaticTriangleGroup`. This object represents a triangle mesh that can collide with entities. It is well suited for creating a physical environment for your game.

To do this, find a model that you want to use and load it into your game in the `LoadContent` method. The information stored in the model needs to be put into a form that `BEPUPhysics` can understand. The `StaticTriangleGroup` offers a helper method for extracting this information:

```
StaticTriangleGroup.StaticTriangleGroupVertex[] vertices;  
int[] indices;  
StaticTriangleGroup.GetVerticesAndIndicesFromModel(PlaygroundModel, out vertices, out indices);
```

The `PlaygroundModel` is an example model used in the `GettingStartedDemo`. Now create a `TriangleMesh` from the vertices and indices:

```
TriangleMesh triangleMesh = new TriangleMesh(vertices, indices);
```

The triangle mesh takes a `TriangleMeshVertex` array as its first parameter and an array of ints as a second parameter. The vertices array created previously was of the type `StaticTriangleGroupVertex`, which is a subclass of `TriangleMeshVertex`. It includes a bit of extra optional information that can be used by the `StaticTriangleGroup`. Now, construct the `StaticTriangleGroup` itself using the `TriangleMesh`:

```
StaticTriangleGroup triangleGroup = new StaticTriangleGroup(triangleMesh);
```

The `StaticTriangleGroup` will query the `TriangleMesh` to manage collisions with entities. Since the `StaticTriangleGroup` starts a bit high relative to the rest of the simulation, modify its `WorldMatrix` to move it down:

```
triangleGroup.worldMatrix = Matrix.CreateTranslation(new Vector3(0, -40, 0));
```

And finally, add the `StaticTriangleGroup` to the space just as we added entities to the space:

```
space.Add(triangleGroup);
```

Try setting up a renderer for the triangle mesh. You can get an example `StaticModel` game component from the `GettingStartedDemo`. Once it's added, you can run the game and see the mesh below the boxes we set up previously. Try shooting some boxes at it and watching them collide.

5 | Handling an Event

One effective way of binding your game's logic to the physics engine is by using events. There are a variety of events that can trigger related to collisions; a listing of them can be found in the `BEPUPhysics.Events` namespace.

To set up an event, use an entity's `EventManager`. Here, we will use the `InitialCollisionDetected` event which fires when the first contact point in a collision pair is created. When the event is triggered, the entity's event manager will call the event handling method that was passed in. Adding the method to the event looks like this:

```
deleterBox.EventManager.InitialCollisionDetected += HandleCollision;
```

The `deleterBox` is another kinematic entity floating near the large 'ground' box. The `HandleCollision` method is:

```
public void HandleCollision(Entity sender, Entity other, CollisionPair pair)
{
    space.Remove(other);
    Components.Remove((EntityModel)other.Tag); //Remove the graphics too.
}
```

The method signature matches the required signature of the `InitialCollisionDetectedEventHandler`. The "sender" parameter is the entity that has the event hook, in this case the `deleterBox`. The "other" entity is the entity colliding with the sender. The `CollisionPair` is the object which oversees the collision between the two entities. A `CollisionPair` exists between any two entities that are in danger of colliding and have overlapping bounding boxes.

This event handler removes the incoming entity from the space and removes the entity's graphics from the game's component list. In the `GettingStartedDemo`, the `EntityModel` rendering object for each entity is put into the entity's tag property. An entity's tag is an object that can be set by the user to store arbitrary data. This tag is referenced by the event handler to retrieve its `EntityModel`.

For more detailed information about events, check the Entity Events documentation on the `BEPUPhysics` website.

6 | Going Further

To learn more about `BEPUPhysics`, check out the other demos and documentation available on the website. Try changing various settings and creating your own simulations.

If you ever need help, please feel free to post on the forums. Asking questions is a great way to learn and helps make `BEPUPhysics` better too!