

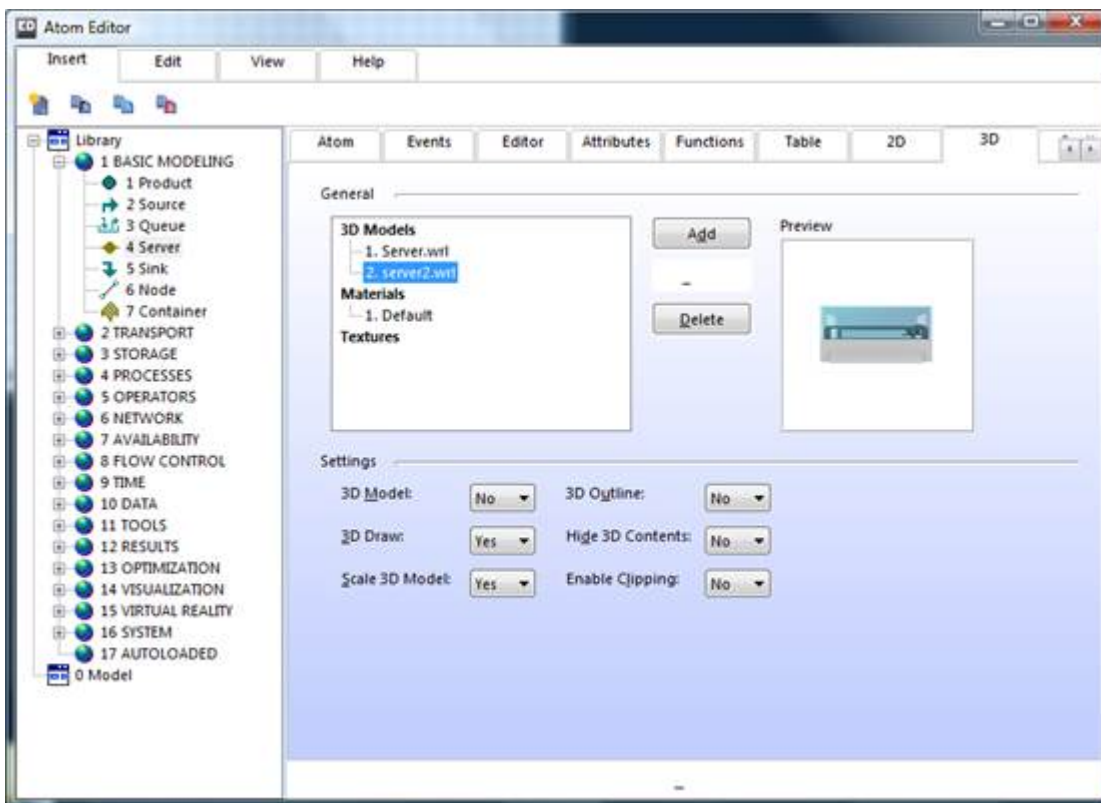
## Quickstart guide to 3D animation

One of the key features of Enterprise Dynamics is the availability of 3D visualization. Each object, or so-called Atom, in Enterprise Dynamics can have a 3D representation that is generated in real-time.

The way your atom looks like in 3D can be completely customized. This quick start will give you some insight in the various ways this can be achieved. To be able to modify or add 3D animation to atoms you need to have the ability to use the Atom Editor.

### Step 1: Understanding the 3D properties of an atom

The behavior and properties of an atom are defined in the Atom Editor. In the 3D section of the properties of an atom you can toggle a number of settings on and off.



#### Setting

#### Description

3D Model

When switched on this setting will display the first 3D model that is attached to the atom. In the above screen shot the atom would display Server.wrl.

3D Draw

This switch determines whether the On3DDraw event handler is executed when an atom needs to be drawn.

Scale 3D Model

When the *3D Model* switch is on at default the 3D model is drawn at its own size. When the switch *Scale 3D Model* is also used the 3D model is scaled to the size of the atom.

3D Outline

This switch is used to display an outline of the atom. The outline is a box of the size of the atom drawn in the layout color that can be set in the 2D section.


Hide 3D Contents

An atom can contain multiple atoms that can be drawn individually. When the *Hide 3D Contents* is switched on these child atoms are not drawn.

Enable Clipping

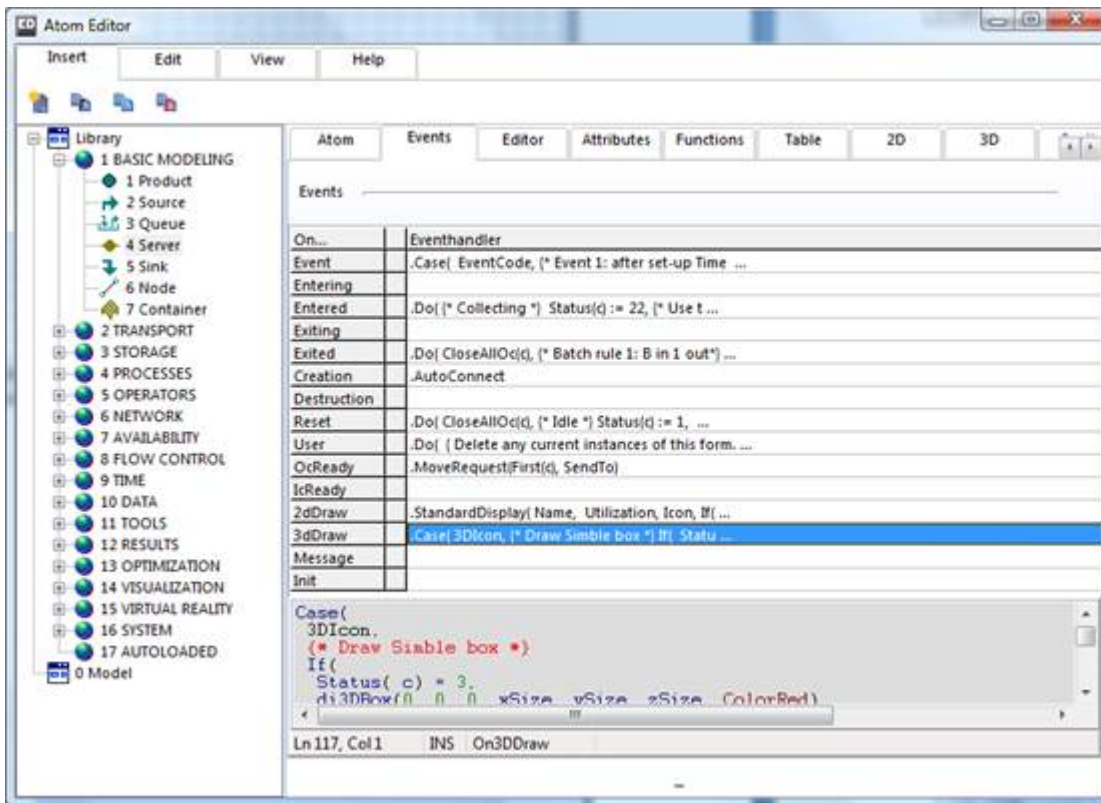
This switch is for future use.

From these settings the *3D Draw* setting is the most important setting if you want to draw your own 3D visualization or you want to control the behavior of imported 3D objects.

 Enterprise Dynamics supports the following 3D objects: VRML 1.0, 2.0, 3D Studio and 3D dxf.

## Step 2: Using the On3DDraw event handler

The On3DDraw event handler gives you full control to the visualization of your atom. By using 4DScript you determine what is drawn. You can enter a single line of code resembling the switches described earlier up to many lines of code to draw complex 3D visualizations.



Enterprise Dynamics contains many 4DScript functions to draw visual elements or objects. You can use them individually or in combination. The functions can be divided into 3 sections:

1. Primitives
2. Objects
3. Coordinate system


With sections 1 and 2 you draw something on the screen based on a x,y, and z-axis. Section 3 enables you to modify the perspective of your coordinate system.

### Section 1: Primitives

Enterprise Dynamics has multiple functions to draw primitives:

- DrawCone
- DrawCube
- DrawCylinder
- DrawPlane
- DrawSphere

You can use these functions to draw primitives to construct a complex object. All the functions contain parameters to set the location and some of the typical aspects of that primitive.

 There are more 3D draw functions available in Enterprise Dynamics than the primitive functions described in this tutorial. You can find these functions in the category Visualization.

**Example 1:**

Drag an atom in your model. Set the 3D settings of this atom in such a way that 3D Draw is switched on while all other switches are switched off. Enter the following code in the On3DDraw event handler:

**On3DDraw: draw a cone**

```
DrawCone(0, 0, 0, 10, 10, 5)
```



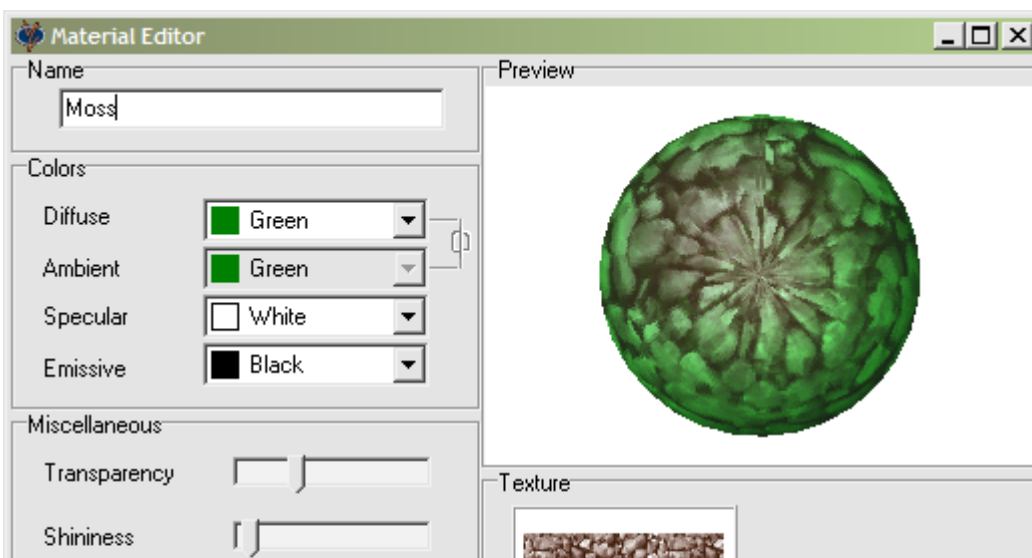
As you can see in the example the cone is colored grey. Actually all the functions to draw primitives do not have a color property. Colors are set via a material that is “glued” to the object.

You can add, edit or remove materials in the 3D properties sheet in the Atom Editor. The materials can be given a name and attached to an atom. In this way you could change material in your code if wanted.

There are various colors to add to the material:

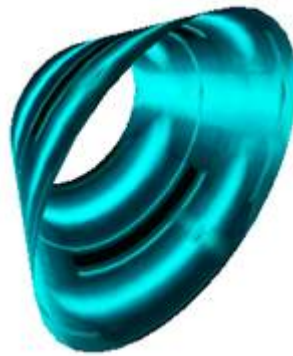
Color	Description
Diffuse	The color from direct light.
Ambient	The color from environmental light.
Specular	The color of viewer eye point caused by reflection.
Emissive	The color the object itself emits.

The diffuse and ambient lights are connected by default but by clicking on the line between these 2 colors you can disconnect them and set them separately.



**Example 2:**

Use the same situation as in the first example. Now modify the default material of the atom. By altering the colors and the texture the same 4DScript code could result in the same cone, but with a different look. `DrawCone(0, 0, 0, 10, 10, 5)`



You could also enter the following code to select the second material attached to your atom:

**Select second material**

```
Do (
  SetCurrentMaterial(Material(2)),
  DrawCone(0, 0, 0, 10, 10, 5)
)
```

In this case a texture of bricks was selected which gives the following cone:



Expand your 4DScript code in any way you like. You can combine primitives and use conditions to change materials during a simulation run.

**Section 2: Objects**

As you have seen in *step 1* Enterprise Dynamics has the ability to visualize 3D objects made with various 3D modeling tools. This section will demonstrate how to control these objects via 4DScript.

There are four advantages of using 4DScript to visualize 3D objects in comparison to use the *3D Model* setting in the 3D property sheet:

- You have more control to visualize the 3D object.
- You can combine multiple 3D objects to be visualized for 1 atom.
- You can change 3D objects during run time.
- You can control individual parts of the 3D object (called meshes).

**Example 3:**

Drag an atom in your model. Select the newly created atom in the Atom Editor and switch the 3D Draw setting to on while switching all other settings to off. The next step is to erase all 4DScript code in the On3DDraw event

handler.

You should now have an atom that will visualize itself in 3D by using the On3DDraw event handler, but displays nothing since it has no visualization code.

Add a 3D object to the atom. In this example we have the box-closed.wrlfile. Enter the following code in the On3DDraw event handler:

### On3DDraw

```
DrawModel3D (Model3D (1))
```

The function `DrawModel3D` visualizes a 3D object. The `Model3D` command is a reference to the 3D object that is attached to the atom. Since the newly added object is at position 1 to the atom you can reference it via 1.



You can also reference 3D objects stored in Enterprise Dynamics by using the function `Model3DByName`. This function obtains a reference to the 3D object based on the given unique name.

Example 3 demonstrates how easy it is to visualize a 3D object in Enterprise Dynamics. However the 3D object is visualized in its own size at its default location (0, 0, 0). By using the optional parameters of the function `DrawModel3D` you can also set the position and the scale of the 3D object:

Parameter	Description
e1	The unique ID of the 3D object you want to visualize. Use the <code>Model3D</code> function to make a reference to 3D objects attached to your atom.
e2	The x-location of the 3D object.
e3	The y-location of the 3D object.
e4	The z-location of the 3D object.
e5	The x-scale of the 3D object.
e6	The y-scale of the 3D object.
e7	The z-scale of the 3D object.

Using 3D objects can be much faster than drawing complex 3D animations based on primitives. This is caused by the fact that the use of primitives requires that each operation is internally interpreted by Enterprise Dynamics and then executed. A 3D object is send directly to the OpenGL engine and requires less internal interpretation time.

#### Example 4:

It is no problem to combine multiple 3D objects in the On3DDraw event handler. Add an additional 3D object to your atom. In this example we added the box-opened.wrl file to the atom. Use the following 4DScript code:

**Visualize two 3D objects**

```
Do (
  DrawModel3D (Model3D(1), 0, 0, 0),
  DrawModel3D (Model3D(2), 2, 0, 0)
)
```

This will visualize both the 3D objects close to each other.



In the same way you could use a condition to switch between different 3D objects during runtime.

Up to now we have used the 3D objects as a single entity to visualize something in 3D. However, when 3D designers develop more complex 3D objects they will use sub objects called meshes. These meshes can be controlled individually by Enterprise Dynamics.

The capability to control these meshes allows you to add movement to your 3D visualization in an easy way. In Enterprise Dynamics we use the 4DScript function [DrawModel3DMesh](#) to achieve this.

**Example 5:**

A default 3D object in Enterprise Dynamics that contains meshes is Server.wrl. This 3D object is used in the standard Server atom and it contains 3 meshes.

If you examine the On3DDraw event handler in the atom you can see that meshes 1 and 3 are used as static 3D visualizations. These 2 meshes visualize the outline of the 3D object. Mesh 2 visualizes the part inside the machine that we want to move depending on a product inside the machine.

Although the code looks more complex than the earlier examples it is actually not that hard to understand. The [DrawModel3DMesh](#) function has the same parameters as the [DrawModel3D](#) function, but also requires an index of the mesh.

```
{ Machine }
Do
(
  { Display the machine. The machine is a VRML object and needs some
    repositioning and scaling. }

  DrawModel3DMesh(model3D(1, c), 1, xSize * 0.5, 0.5, 0, xSize * 0.25, ySize, 1.2 * zSize),
  DrawModel3DMesh(model3D(1, c), 3, xSize * 0.5, 0.5, 0, xSize * 0.25, ySize, 1.2 * zSize),
  var([valOffset], vbValue, xsize * -0.19),
  var([valInter], vbValue),
  if(CurCycle = 0,
    valInter := 0,
    valInter := Min(1, (Time - BeginBusy) / CurCycle)
  ),
  if(AtomExists(First(c)),
    SetLoc(
      (xsize * -0.15 + (xsize(First(c)) * 0.5) +
        valInter * (xsize(c) * 0.35 - (xsize(First(c))) * 0.5)),
      yLoc(First(c)),
      zLoc(First(c)),
      First(c)
    )
  ),
  if(Status(c) <> 2,
    if(CurCycle = 0,
      valInter := 0,
      valInter := Max(0, -1 * (Time - EndBusy) / CurCycle + 1)
    )
  ),
  DrawModel3DMesh(model3D(1, c), 2,
    xSize(c) * 0.5 + valOffset + valInter * xsize(c) * 0.35, 0.5, 0,
    xSize(c) * 0.25, ySize(c), 1.2 * zSize(c)
  )
)
```

All other code is to take the size of the atom into account and the cycle time of the atom. During the cycle time the product atom needs to be moved from the entry point of the server atom to the exit point of the server atom.

By using `DrawModel3DMesh` it is possible to visualize movement within a 3D object which gives your visualization a more realistic look.

**✍ Before loading a 3D object into Enterprise Dynamics, there are a few guidelines to follow:**

1. Always make sure that the object is mirrored. In Enterprise Dynamics the object is mirrored and loading a non-mirrored object will therefore cause mirrored text etc.
2. Position your drawing (after mirroring) in the upper right quadrant. This allows a correct positioning when loading the object in Enterprise Dynamics.
3. When an object needs to be scaled, rule 2 doesn't apply. These objects must be centered on the origin. Otherwise the scaled object in Enterprise Dynamics will also be moved, since it will scale based on the position relative to the origin. Positioning the objects centre on the origin prevents these movements. This part of the 3D object must then be positioned within Enterprise Dynamics onto its wanted place.
4. In Enterprise Dynamics, when working with different meshes (for moving animation reasons), always check the order in which the meshes are drawn. During converting from one format to another the order can change.

### **Step 3: Coordinate system**

When a primitive or 3D object is visualized it is drawn in the standard Enterprise Dynamics coordinate system. However quite often you want to rotate or translate visualizations. Enterprise Dynamics offers 2 functions to realize that:

- `RotateCoords`
- `TranslateCoords`

An additional 2 functions enables you to save the current coordinate system before you modify the coordinate system or to retrieve the stored coordinate system:

- `PushCoords`
- `PopCoords`

#### **Example 6:**

Start with the earlier example in which you draw a cone. Replace the code with the following 4DScript code:

#### **Draw cone in normal and rotated and translated coordinate system**

```
Do (  
  DrawCone(0, 0, 0, 10, 10, 5),  
  PushCoords,  
  RotateCoords(45, 0, 0, 1),  
  TranslateCoords(15, 15, 0),  
  DrawCone(0, 0, 0, 10, 10, 5),  
  PopCoords  
)
```

You would now see 2 cones. The original cone and a cone rotated 45 degrees over the z-axis and translated over the x and y-axes.



The [PushCoords](#) and [PopCoords](#) functions ensure that the original coordinate system is stored at a stack and is reset after the rotated and translated cone is drawn. In functions in which you add even more visualization this would allow you various rotations and translations while maintaining the original coordinate system.

When multiple coordinate systems are pushed they are stored in order. So the first time you will use [PopCoords](#) it will reset the last coordinate system that was pushed with [PushCoords](#) and so on.