



TWaver[®] 3D for Flex

Developer Guide

Version 3.0

Feb 2013

Serva Software

info@servasoftware.com

<http://www.servasoftware.com>

PO Box 8143, Wichita Falls, Texas, USA 76307



For more information about Serva Software and TWaver please visit the web site at:

<http://www.servasoftware.com>

Or send e-mail to:

info@servasoftware.com

Feb, 2013


Notice:

This document contains proprietary information of Serva Software. Possession and use of this document shall be strictly in accordance with a license agreement between the user and Serva Software, and receipt or possession of this document does not convey any rights to reproduce or disclose its contents, or to manufacture, use, or sell anything it may describe. It may not be reproduced, disclosed, or used by others without specific written authorization of Serva Software.

TWaver, servasoft, Serva Software and the logo are registered trademarks of Serva Software. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S.A. and other countries. Other company, brand, or product names are trademarks or registered trademarks of their respective holders. The information contained in this document is subject to change without notice at the discretion of Serva Software.

Copyright © 2013 Serva Software LLC
All Rights Reserved

Table of Contents

- Home 
 - Preface
 - Building your first 3D application
 - Flex Builder 4.6 As Development Tools
 - Create first application
 - Structs
 - Event-driven
 - Element
 - ElementBox
 - Network3D
 - 3D Styles
 - Primitive Objects
 - Camera
 - Common Camera
 - Orbit or Hover Camera
 - Coordinate System
 - World Space
 - Parent Space
 - Local Space
 - Position 3D Objects
 - Rotate 3D Objects
 - Scale 3D Objects
 - Materials
 - Bitmap Materials
 - Color Material
 - Only Color
 - Append Color
 - Alpha Effect
 - Model File
 - Import External Models
 - Load DAE models
 - Load 3DS models
 - Load Obj models
 - Load md2 models
 - TWaver Format Model File
 - Resource Management
 - Embed Resources into Project
 - Access Resources at Runtime
 - Interactions
 - Use camera
 - Mouse Interaction
 - Use other Events
 - Use Camera3DEvent
 - Use ModelLoadEvent

- Effect Functions On Network3D
 - SelectedEffectFunction
 - SelectFunction
 - SelectedColorFunction
 - SelectedWidthFunction
 - SelectedGlowFunction
 - SelectedBlurFunction
 - SelectedScaleFunction
- Render Alarms
 - AlarmEnableFunction
 - AlarmColorFunction

Home

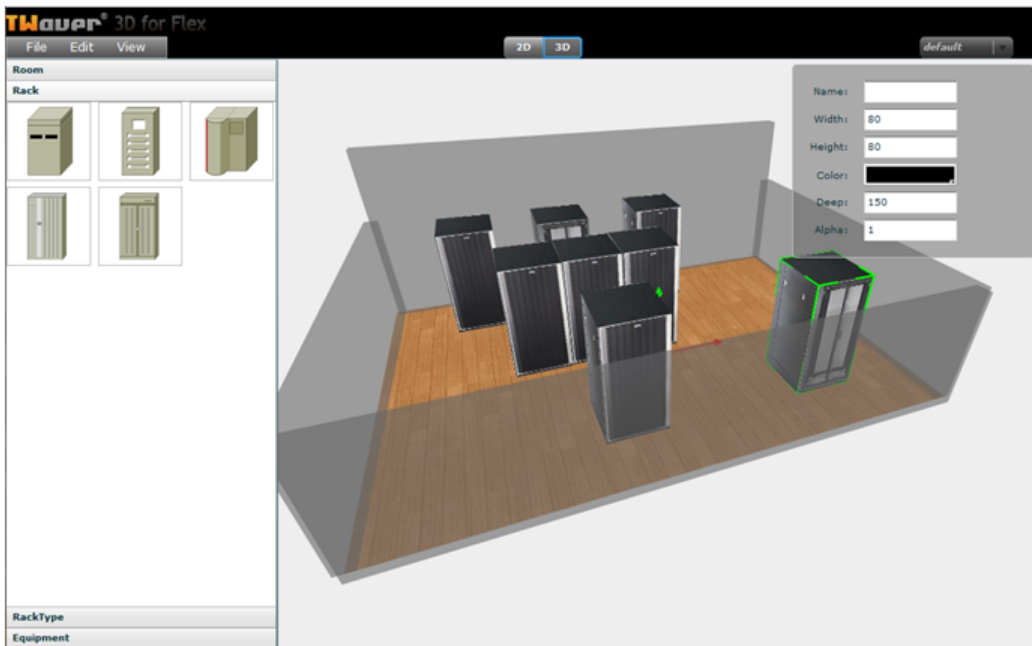
This Document will introduce how to use TWaver 3D for Flex.

- [Preface](#)
- [Building your first 3D application](#)
- [Structs](#)
- [Primitive Objects](#)
- [Camera](#)
- [Coordinate System](#)
- [Materials](#)
- [Model File](#)
- [Resource Management](#)
- [Interactions](#)
- [Effect Functions On Network3D](#)
- [Render Alarms](#)

Preface

TWaver 3D for Flex is a toolkit which helps programmers implement 3D scene in their application. It allows them to create a wide range of 3D applications, including 3D rooms, detailed 3D chassis, and even 3D topology.

TWaver 3D for Flex is based on Flex Framework and supports Flex SDK4.5.1 and the later versions. This toolkit provides some 3D primitive objects, such as the cube, cone, sphere, straight line, etc. It also provides some interaction modes for developers to make good interactive applications, such as rolling a 3D scene, selecting an object by clicking it, touring in the scene, displaying the scene in First or Third Person.



Building your first 3D application

To use TWaver3D, you need TWaver.swc library and TWaver3D.swc library which can be downloaded from Serva Software website www.servasoftware.com. Also, you need an IDE. From version 3.0, TWaver 3D supports Stage3D, and Flash Builder 4.6 is a best choice. You may would want to open source IDE as you do FlashDevelop and this is ok. The following instruction will explain how to use TWaver.swc library and TWaver3D.swc library in Flex Builder 4.6.

If developers choose other IDEs, please choose Flex SDK4.6 and Flash player 11 or above.

- [Flex Builder 4.6 As Development Tools](#)
- [Create first application](#)

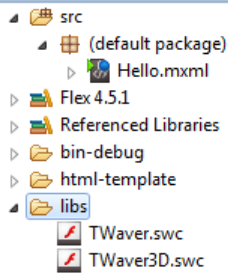
Flex Builder 4.6 As Development Tools

Flex Builder 4.6 is already accompanied by Flex 4.6 SDK and Debug Flash Player 11, so it is unnecessary for users to install them otherwise.

Create first application

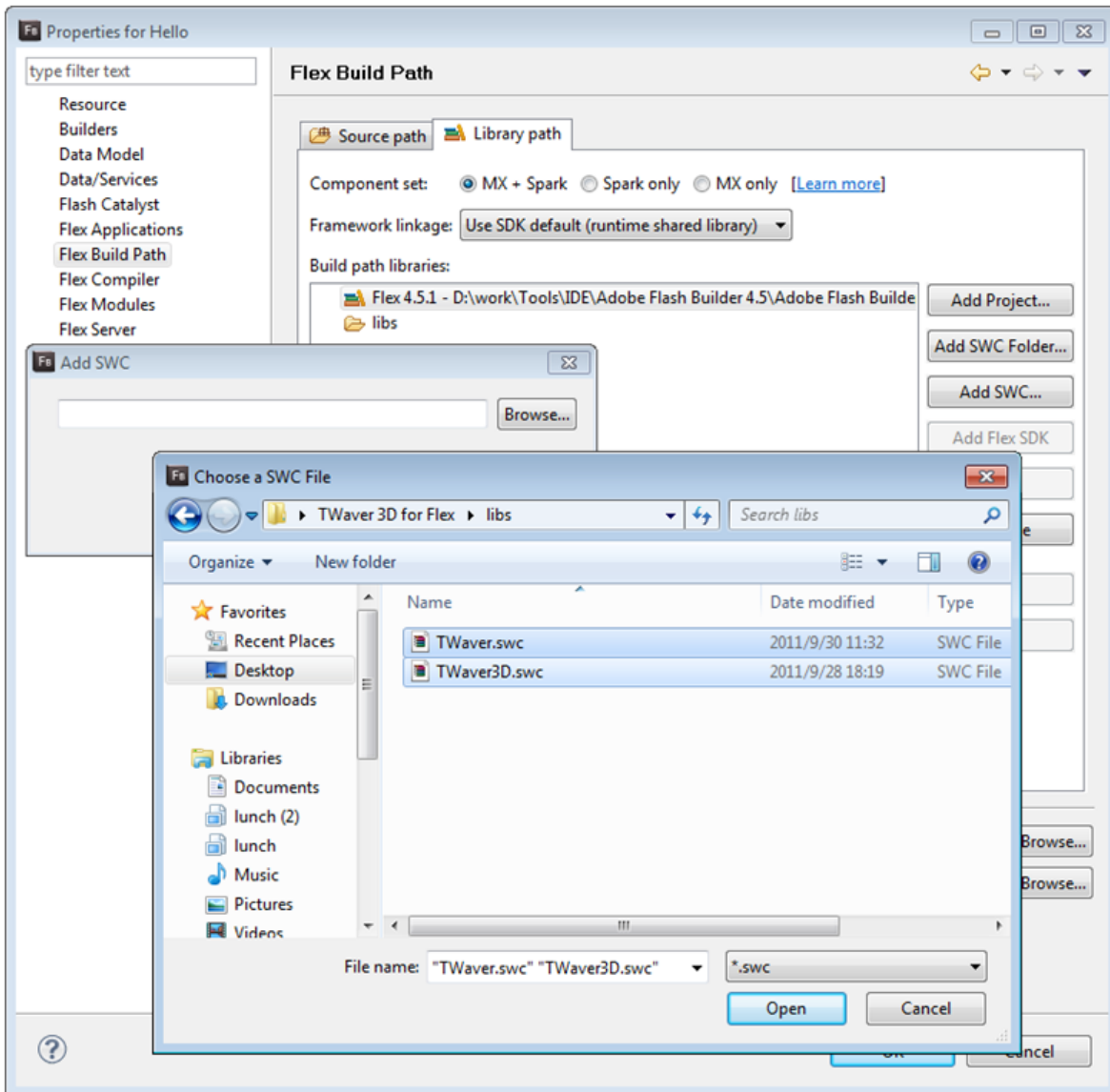
Create a New Project

You can create a new project HelloTWaver3D by following the step: File -> New -> Flex Project and then put TWaver.swc and TWaver3D.swc file in libs directory. Flex Builder4.5 will add all *.swc files under libs directory to compile class library automatically.



Add TWaver.swc to build path

Users can add TWaver.swc and TWaver3D.swc to build path as below, add other library files, and use other Flex SDK versions.



Setup Flex SDK and Debug Flash Player

References:

<http://www.adobe.com/products/flex/>

<http://www.adobe.com/support/flashplayer/downloads.html>

Create first application

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600" backgroundAlpha="0"
    xmlns:components="components.*">
    <components:RoomView id="view" width="100%" height="100%" />
</s:Application>
```

```

package components
{
    import flash.events.MouseEvent;
    import flash.geom.Point;
    import flash.geom.Vector3D;

    import mx.controls.Alert;

    import spark.components.Group;

    import twaver.*;
    import twaver.threed.*;
    import twaver.threed.event.Mouse3DEvent;
    import twaver.threed.network.Network3D;
    import twaver.threed.util.Consts3D;
    import twaver.threed.util.Style3D;
    import twaver.threed.util.Util3D;

    public class RoomView extends Group
    {
        private var box:ElementBox=new ElementBox();
        private var network3d:Network3D;
        [Embed(source="images/machine.png")]
        private static const R:Class;
        [Embed(source="images/floor.jpeg")]
        private static const Floor:Class;
        public function RoomView()
        {
            super();
            Util3D.registVector3D();
            Utils.registerImageByClass("rackDImage1",R);
            Utils.registerImageByClass("floor",Floor);
        }

        override protected function createChildren():void
        {
            super.createChildren();
            network3d=new Network3D(box);
            initData();
            initAlarm();
            network3d.showAxes();
            network3d.applyHoverCamera(-180, 5, 1600);
            network3d.tiltAngleLowLimit=-80;
            network3d.tiltAngleUpLimit=80
            network3d.percentWidth=100;
            network3d.percentHeight=100;
            network3d.backgroundColor = 0x925618;

            this.addElement(network3d);
        }

        private function initData():void
        {
            initRoom();
            initRack();
        }

        private function initAlarm():void
        {
            var e:IElement=box.getElementByID("0:1");
            var alarm:Alarm=new Alarm(null, e.id, AlarmSeverity.CRITICAL);
            box.alarmBox.add(alarm);
            e=box.getElementByID("2:2");
        }
    }
}

```

```

        alarm=new Alarm(null, e.id, AlarmSeverity.MAJOR);
        box.alarmBox.add(alarm);
        e=box.getElementByID("1:0");
        e.setStyle(Style3D.MATERIAL_COLOR,0x00ff00);
    }

    private function initRack():void
    {
        var xgap:Number=size / 4;
        var ygap:Number=size / 4;
        var startX:Number=-xgap;
        var startY:Number=-ygap;
        for (var i:int=0; i < 3; i++)
        {
            for (var j:int=0; j < 3; j++)
            {
                box.add(createRack(i + ":" + j, startX + i * xgap, startY + j * xgap));
            }
        }
    }

    /**
     *
     * @param x centerX
     * @param z centerZ
     */
    private function createRack(id:String, x:Number, z:Number):IElement
    {
        var element:IElement=new Element(id);
        element.setStyle(Style3D.MAPPINGTYPE, Consts3D.MAPPINGTYPE_MAP6);
        element.setStyle(Style3D.MAPPING_COMMON_PATH, "rackDImage1");
        element.setStyle(Style3D.PROPERTY_SIZE, new Vector3D(80, 150, 80));
        element.setStyle(Style3D.PROPERTY_SPACE_LOCATION, new Vector3D(x, 150 / 2 + 10, z));
        return element;
    }

    private function initRoom():void
    {
        buildWalls();
        buildFloor();
    }

    private var size:Number=800;

    private function buildWalls():void
    {
        box.add(createWall(0, size / 2, size, 0));
        box.add(createWall(0, -size / 2, size, 0));
        box.add(createWall(-size / 2, 0, size, 90));
        box.add(createWall(size / 2, 0, size, 90));
    }

    private function buildFloor():void
    {
        var element:IElement=new Element();
        element.setStyle(Style3D.MAPPINGTYPE, Consts3D.MAPPINGTYPE_COMMON);
        element.setStyle(Style3D.THREED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_PLANE);
        element.setStyle(Style3D.PROPERTY_SMOOTH_LEVEL, Consts3D.SMOOTH_LEVEL_GREAT);
        element.setStyle(Style3D.WITH_TILED_MATERIAL, true);
        element.setStyle(Style3D.MAPPING_COMMON_PATH, "floor");
        element.setStyle(Style3D.PROPERTY_SPACE_LOCATION, new Vector3D(0, 0, 0));
        element.setStyle(Style3D.PROPERTY_SIZE, new Vector3D(size + 10, 0, size + 10));
        box.add(element);
    }

    /**
     *

```

```

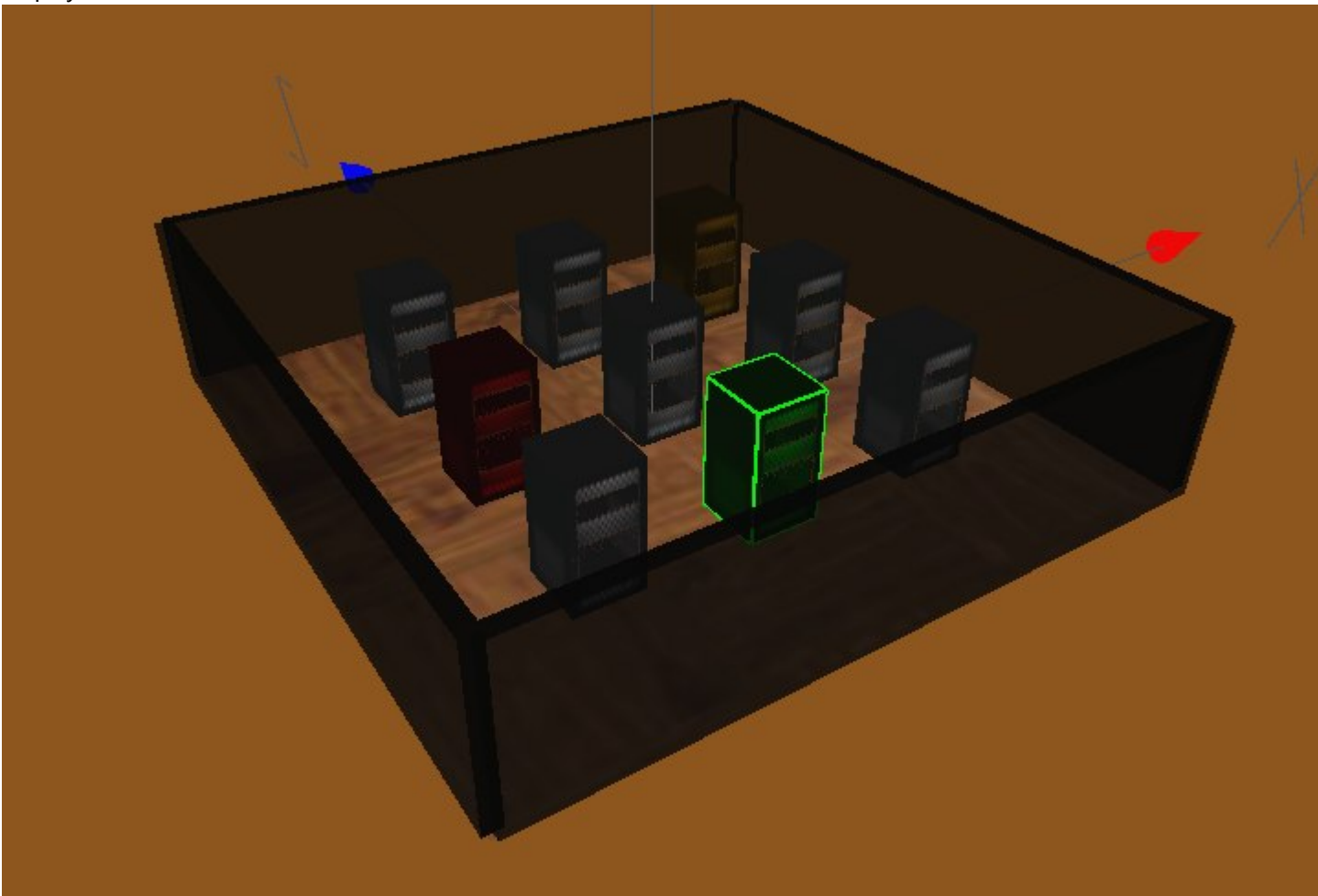
*      | z |
*      |   |
*      | . x|
*      |(0,0)|
*      |   |
*      |   |
*      |-----|
**/
private function createWall(x:Number, z:Number, width:Number, angle:Number=0):IElement
{
    var wall:IElement=new Element();
    wall.setStyle(Style3D.MATERIAL_COLOR, 0XCCCCCC);
    wall.setStyle(Style3D.THREED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_CUBE);
    wall.setStyle(Style3D.MAPPINGTYPE, Consts3D.MAPPINGTYPE_COLOR);
    wall.setStyle(Style3D.MATERIAL_ALPHA, 0.8);

    wall.setStyle(Style3D.SELECTED_EFFECT_COLOR, 0x00ff00);

    wall.setStyle(Style3D.PROPERTY_SIZE, new Vector3D(width, 200, 20));
    wall.setStyle(Style3D.PROPERTY_SPACE_LOCATION, new Vector3D(x, 200 / 2, z));
    wall.setStyle(Style3D.PROPERTY_ROT_ANGLE, new Vector3D(0, angle, 0));
    return wall;
}
}

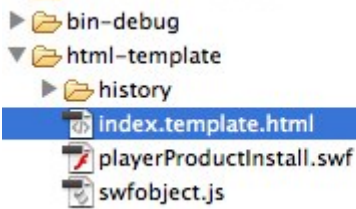
```

Displayed as below:



Enable Stage3D work

Find the index.template.html, and setup Direct mode.



Add params.wmode into the file.

```
<script type="text/javascript">
    // For version detection, set to min. required Flash Player version, or 0 (or 0.0.0), for no version detection.
    var swfVersionStr = "${version_major}.${version_minor}.${version_revision}";
    // To use express install, set to playerProductInstall.swf, otherwise the empty string.
    var xiSwfUrlStr = "${expressInstallSwf}";
    var flashvars = {};
    var params = {};
    params.quality = "high";
    params.bgcolor = "${bgcolor}";
    params.allowscriptaccess = "sameDomain";
    params.allowfullscreen = "true";
    params.wmode = "direct";
    var attributes = {};
    attributes.id = "${application}";
    attributes.name = "${application}";
    attributes.align = "middle";
    swfobject.embedSWF(
        "${swf}.swf", "flashContent",
        "${width}", "${height}",
        swfVersionStr, xiSwfUrlStr,
        flashvars, params, attributes);
    // JavaScript enabled so display the flashContent div in case it is not replaced with a swf object.
    swfobject.createCSS("#flashContent", "display:block;text-align:left;");
</script>
```

Structs

TWaver 3D for Flex is developed based on TWaver Flex 2.0. Every 3D object is described by TWaver. Element and 3D scene is displayed in a visual component called Network3D. If the specified 3D styles are assigned to an Element instance which is inserted into a TWaver's data container(ElementBox), it will be displayed in a Network3D component which is binding to that data container. In order to provide good interactions between the 3D scene and the end users, TWaver 3D for Flex has built flexible interaction mechanism and predefined several basic interactions, such as selecting an element with a click on the 3D object, clearing all selections by double click on the scene, and moving in the scene with direction key, etc.



Event-driven

As the other products in TWaver, TWaver 3D for Flex depends on event driving. Developers can add event listeners to the target if they want to control a flow of their program. TWaver 3D can fire 3 kinds of events: Camera3DEvent, ModelLoadEvent and Mouse3DEvent.

To improve the interactions between users and the program, developers can flexibly use the listeners of those events.

1. Monitor the state of the camera

When the state of the camera is changed, Camera3DEvent will occur. Developers can add listeners to a Network3D component to receive this kind of event.

```
network.addEventListener(Camera3DEvent.CAMERA_UPDATE,onCameraStateChanged);

function onCameraStateChanged(evt:Camere3DEvent):void{
    //camera's direction
    var direction:Vector3D = network.cameraDirection

    //camera's position
    var pos:Vector3D = value.position;

    //you can do more with the camera's states,...
}
```

2. Increase mouse interactions

Network3D is a component which extends UIComponent. Developers can add mouse or keyboard event listeners to a Network3D component by using addEventListener. In order to control the interaction between mouse and the 3D scene, developers have to use addMouseOnElementListener or addMouseOnCanvas. For more details of this part, please refer to [Mouse Interaction](#).

3. Monitor the loading progress when loading external models

When loading external models from remote servers, developers often want to know the progress of the loading. When the loading is finished, developers will repaint the scene. If the loading fails, developers will give some advice to the users.


```
Util3D.addEventListener(ModelLoadEvent.LOAD_SUCCESS,
    function(evt:ModelLoadEvent):void{
        //do what you want after the model file is loaded into the scene successfully.
    });
```

or

```
Util3D.addEventListener(ModelLoadEvent.LOAD_ERROR,
    function(evt:ModelLoadEvent):void{
        //give users tips about failure
    });
```


Element

The 3D model is defined by Element Class which is the basic data model in TWaver. When assigning appropriate 3D style values to an Element object, Network3D will build a 3D model in the 3D scene. All the styles used to describe the 3D properties of an element are defined in `twaver.threed.utils.Style3D`. The styles include `PROPERTY_SIZE`, `PROPERTY_SPACE_LOCATION`, `PROPERTY_ROT_ANGLE`, `MATERIAL_COLOR`, `MATERIAL_ALPHA`, and `MAPPINGTYPE`, etc.

 Note: For more details about Style3D, you can refer to [3D Styles](#).

ElementBox

ElementBox is the default data container of the Element objects. The container is designed to manage the Element objects, to monitor the changes in the properties of Element objects, and to drive the visual components in TWaver.

Network3D

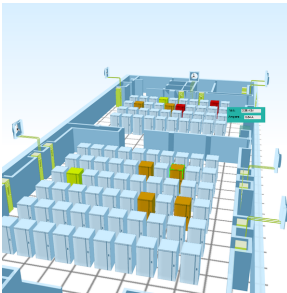
Network3D is an interactive visual component which extends UIComponent. It is designed to display the 3D scene and to implement the interaction between users and the 3D scene. The Network3D component only displays the models which are put into the related ElementBox. An ElementBox instance is binding to a Network3D object as default when the Network3D object is initialized. So if you want a Network3D object to be binding to another ElementBox, you can use the code as below:

```
network.elementBox = otherBox;
```

Network3D represents the 3D scene, and renders the 3D objects with textures and/or colors. It provides interaction between users and 3D scene. Users can select a 3D object, append color to its surface, rotate the scene, and even move the camera in the scene. Network3D enables developers to customize the default renderer mechanism by setting up the related function interface. For example, developers can reset Network3D#alarmColorFunction to customize a new mechanism, rendering 3D objects with their own rules.



For more details on customizing renderer mechanism, please refer to [Effect Functions On Network3D](#) and [Render Alarms](#)



3D Styles

All the 3D styles are defined in Style3D class. In order to build a 3D scene, developers can create some element objects and assign necessary style values to the element objects. For example, if you want to display an Element object as a cube in the network, you can use the following code:

```
var cube:Node = new Node();
cube.setStyle(Style3D.THREED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_CUBE);
var box:ElementBox = new ElementBox();
var network:Network3D = new Network3D(box);
box.add(cube);
```

Styles used to describe primitives

3D Style	3D Style Value	Description
Style3D.THREED_SHAPE_TYPE	valid values are listed in Consts3D Consts3D.THREED_SHAPE_TYPE_CUBE Consts3D.THREED_SHAPE_TYPE_SPHERE Consts3D.THREED_SHAPE_TYPE_CONE Consts3D.THREED_SHAPE_TYPE_CYLINDER Consts3D.THREED_SHAPE_TYPE_ROUNDEDCUBE Consts3D.THREED_SHAPE_TYPE_PLANE Consts3D.THREED_SHAPE_TYPE_LINE Consts3D.THREED_SHAPE_TYPE_BILLBOARD Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL Consts3D.THREED_SHAPE_TYPE_LINEAREXTRUSION Consts3D.THREED_SHAPE_TYPE_BOXSCENE Consts3D.THREED_SHAPE_TYPE_COMPLEXCUBE THREED_SHAPE_TYPE_LATHEEXTRUSION Consts3D.THREED_SHAPE_TYPE_TEXT	It is used to describe which kind of primitive is used to represent the specified Element.
Style3D.PROPERTY_SPACE_LOCATION	a Vector3D instance e.g.new Vector3D(100,10,30)	It is used to define the position of the object in the 3D scene. e.g.the object is positioned at the point whose coordinate is (x=100,y=10,z=30)
Style3D.PROPERTY_SIZE	a Vector3D instance e.g.new Vector3D(1000,300,200)	It is used to define the size of the object. e.g.set the object's length/height/depth as 1000/300/200
Style3D.PROPERTY_SCALE	a Vector3D instance or a number e.g.new Vector3D(1,0.5,2) or 3	It is used to stretch out the element. e.g.stretching out the element : kept still along x axis, 0.5 times along y axis, twice along z axis or 3 times along every axis
Style3D.PROPERTY_ROT_ANGLE	a Vector3D instance	It is used to define the rotation angle of an element.
Style3D.PROPERTY_SMOOTH_LEVEL	valid values are Consts3D.SMOOTH_LEVEL_COMMON Consts3D.SMOOTH_LEVEL_MIDDLE Consts3D.SMOOTH_LEVEL_GREAT	It is used to add segments to the facets of an element. For example, in general, a cube is composed of 6 facets and each

		of them has 1*1 segments. If you choose SMOOTH_LEVEL_MIDDLE, each facet will have 24*12 segments; if you choose SMOOTH_LEVEL_GREAT, each facet will have 32*32 segments. It means that each facet will have more meshes and can be represented better.
Style3D.BOTH_SIDES_VISIBLE	a boolean value	It is used to define both sides of a facet. This style has effects on cube.
Style3D.SELECTED_EFFECT_TYPE	valid values are listed as below: Consts3D.EFFECT_TYPE_NONE Consts3D.EFFECT_TYPE_SCALE Consts3D.EFFECT_TYPE_FRAMEWORK	It is used to define the special effect when the element is selected. Default: EFFECT_TYPE_FRAMEWORK. EFFECT_TYPE_GLOW/ EFFECT_TYPE_BLUR works only when ZORDERING_USINGCANVAS is set as true.
Style3D.EFFECT_SCALE_VALUE	number	It defines how many times the element has stretched out when the element is selected and works only when EFFECT_TYPE_SCALE is set up.
Style3D.SELECTED_EFFECT_COLOR	uint	It defines the color of the framework of the selected element and works only when EFFECT_TYPE_FRAMEWORK is set up.
Style3D.SELECTED_FRAMEWORK_WIDTH	number	It defines the width of the lines constituting the framework of the selected element and works only when EFFECT_TYPE_FRAMEWORK is set up.
Style3D.LINE_SHAPE_POINTS	an array filled with Vector3D instances	It defines the shape of a line and works only when THREEED_SHAPE_TYPE_LINE is set up.
Style3D.LINE_SHAPE_WIDTH	number	It defines the width of a line and works only when THREEED_SHAPE_TYPE_LINE is set up.
Style3D.CYLINDER_OPEN	boolean	It defines whether the cylinder has top and bottom and works only when THREEED_SHAPE_TYPE_CYLINDER is set up.
Style3D.ZORDERING_LAYER	int	It defines the the id of the layer at which an element is. The smaller the figure of the id of the layer is, the closer the element is to the camera when TWaver 3D displays the 3D scene. This property can be used to avoid

		displaying 3d objects in wrong layer relationships between different facets.
Style3D.ZORDERING_USINGCANVAS	boolean	Default:false. Makes sure that the element is rendered in its own paint session. This property should not be used with ZORDERING_LAYER together.
Style3D.TEXT_FONT_3D	String	It refers to the name of the font to use and works only when THREEED_SHAPE_TYPE_TEXT is set up.
Style3D.TEXT_FONT_SIZE	number	It defines the size of the text with pixels. Default:20. It works only when THREEED_SHAPE_TYPE_TEXT is set up.
Style3D.TEXT_WIDTH	number	It refers to the width of the drawing area. If the text is longer than this number, then TWaver will start to wrap it. If you do not want the newlines, set this number as Number.POSITIVE_INFINITY. It works only when THREEED_SHAPE_TYPE_TEXT is set up.
Style3D.TEXT_ALIGN	String	It specifies the alignment to the x and y property and works only when THREEED_SHAPE_TYPE_TEXT is set up.
Style3D.TEXT_CONTENT	String	It defines what should be written in the 3D scene and works only when THREEED_SHAPE_TYPE_TEXT is set up.
Style3D.PIN_NEEDLE_LENGTH	number	It defines the length of the needle of the pin object and works only when THREEED_SHAPE_TYPE_PIN is set up.
Style3D.PIN_NEEDLE_COLOR	uint	It defines the color of needle of the pin object and works only when THREEED_SHAPE_TYPE_PIN is set up.
Style3D.PIN_NEEDLE_WIDTH	number	It defines the width of the needle of the pin object and works only when THREEED_SHAPE_TYPE_PIN is set up.
Style3D.PIN_NEEDLE_ALPHA	number. $0 \leq \alpha \leq 1$	It defines the transparency of the the needle of the pin object and works only when THREEED_SHAPE_TYPE_PIN is set up.
Style3D.PIN_HEAD_COLOR	uint	It defines the color of the of a pin object and works only when

		THREED_SHAPE_TYPE_PIN is set up.
Style3D.PIN_HEAD_SIZE	number	It defines the size of head of the pin object and works only when THREED_SHAPE_TYPE_PIN is set up.
Style3D.PIN_HEAD_ALPHA	uint	It defines the transparency of the head of the pin object and works only when THREED_SHAPE_TYPE_PIN is set up.
Style3D.PIN_HEAD_TEXTURE	String of BitmapData	It defines the texture of the head of the pin object and works only when THREED_SHAPE_TYPE_PIN is set up.
Style3D.LINE_SHAPE_POINTS	an Array filled with Vector3D instances	It defines the track of the line and works only when THREED_SHAPE_TYPE_LINE is set up.
Style3D.LINE_SHAPE_WIDTH	number	It defines the width of a line and works only when THREED_SHAPE_TYPE_LINE is set up.

Styles used to describe material

3D Style	3D Style Value	Description
Style3D.MAPPINGTYPE	valid values are listed as below: Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON Consts3D.MAPPINGTYPE_MAP6	It defines which kind of material is used to represent the 3D object. MAPPINGTYPE_MAP6 only supports cube and rounded cube. The line object only supports MAPPINGTYPE_COLOR.
Style3D.MAPPING_COMMON_PATH	String or BitmapData	It defines the source of the texture and works only when MAPPINGTYPE_COMMON is set up. This parameter applies to Consts3D.THREED_SHAPE_TYPE_SPHERE, Consts3D.THREED_SHAPE_TYPE_CONE, Consts3D.THREED_SHAPE_TYPE_CYLINDER, Consts3D.THREED_SHAPE_TYPE_BILLBOARD, Consts3D.THREED_SHAPE_TYPE_BOXSCENE, Consts3D.THREED_SHAPE_TYPE_SPHERESCENE
Style3D.MAPPING_CUBE_LEFT	String or BitmapData	It defines the source of the texture of the left facet. This parameter applies to cube, rounded cube and lathe extrusion.
Style3D.MAPPING_CUBE_FRONT	String or BitmapData	It defines the source of the texture of the front facet. This parameter applies to cube, rounded cube and lathe extrusion.

Style3D.MAPPING_CUBE_RIGHT	String or BitmapData	It defines the source of the texture of the right facet. This parameter applies to cube, rounded cube and lathe extrusion.
Style3D.MAPPING_CUBE_BACK	String or BitmapData	It defines the source of the texture of the back facet. This parameter applies to cube, rounded cube and lathe extrusion.
Style3D.MAPPING_CUBE_TOP	String or BitmapData	It defines the source of the texture of the top facet. This parameter applies to cube, rounded cube and lathe extrusion.
Style3D.MAPPING_CUBE_BOTTOM	String or BitmapData	It defines the source of the texture of the bottom facet. This parameter applies to cube, rounded cube and lathe extrusion.
Style3D.WITH_TILED_MATERIAL	boolean	It defines whether the texture should be tiled on the surface of the specified element.
Style3D.MAPPING_CUBE_LEFT_TILED	boolean	It defines whether the texture should be tiled on the left facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_FRONT_TILED	boolean	It defines whether the texture should be tiled on the front facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_RIGHT_TILED	boolean	It defines whether the texture should be tiled on the right facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_BACK_TILED	boolean	It defines whether the texture should be tiled on the back facet of the specified element

		and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_TOP_TILED	boolean	It defines whether the texture should be tiled on the top facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_BOTTOM_TILED	boolean	It defines whether the texture should be tiled on the bottom facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_LEFT_COLOR	uint	It defines the color of the left facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_FRONT_COLOR	uint	It defines the color of the front facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_RIGHT_COLOR	uint	It defines the color of the right facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH

		should be set as null at the same time.
Style3D.MAPPING_CUBE_BACK_COLOR	uint	It defines the color of the back facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_TOP_COLOR	uint	It defines the color of the top facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_BOTTOM_COLOR	uint	It defines the color of the bottom facet of the specified element and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_LEFT_ALPHA	number, 0<=alpha<=1	It defines the transparency of the material of the left facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_FRONT_ALPHA	number, 0<=alpha<=1	It defines the transparency of the material of the front facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_RIGHT_ALPHA	number, 0<=alpha<=1	It defines the transparency of the material of the right facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON.


		If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_BACK_ALPHA	Number, 0<=alpha<=1	It defines the transparency of the material of the back facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_TOP_ALPHA	number, 0<=alpha<=1	It defines the transparency of the material of the top facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_CUBE_BOTTOM_ALPHA	Number, 0<=alpha<=1	It defines the transparency of the material of the bottom facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_LEFT_TYPE	valid values are listed as below: Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON	It defines the mapping type of the left face and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_FRONT_TYPE	valid values are listed as below: Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON	It defines the mapping type of the front facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_RIGHT_TYPE	valid values are listed as below: Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON	It defines the mapping type of the right facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as

		Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_BACK_TYPE	valid values are listed as below: Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON	It defines the mapping type of the back facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_TOP_TYPE	valid values are listed as below: Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON	It defines the mapping type of the top facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.
Style3D.MAPPING_BOTTOM_TYPE	valid values are listed as below: Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON	It defines the mapping type of the bottom facet and works only when the MAPPINGTYPE of a cube/rounded cube/lathe extrusion is set as Consts3D.MAPPINGTYPE_COMMON. If the element is a lathe extrusion, its MAPPING_COMMON_PATH should be set as null at the same time.

Styles used to control alarm attachment

3D Style	3D Style values	Description
Style3D.ALARM_RENDER_ENABLE	boolean	It defines whether the 3d object is presented with alarm information. For example, whether the element's texture is dyed with the alarm color, or displayed with a bubble.
Style3D.ALARM_ELEMENTDYE_ENABLE	boolean	It defines whether the 3d object's texture can be dyed with the alarm color and works only when ALARM_RENDER_ENABLE is set as true. The color value is controlled by Network3D#alarmColorFunction. Default: the highest severity's color of the new occurring alarms.
Style3D.ALARM_BUBBLE_ENABLE	boolean	It defines whether the 3d object has a alarm bubble and works only when

		ALARM_RENDER_ENABLE is set as true.
Style3D.ALARMATTACHMENT_SHAPETYPE	<p>Values are listed in Consts3D</p> <p>Consts3D.THREED_SHAPE_TYPE_CUBE, Consts3D.THREED_SHAPE_TYPE_SPHERE, Consts3D.THREED_SHAPE_TYPE_CONE Consts3D.THREED_SHAPE_TYPE_CYLINDER Consts3D.THREED_SHAPE_TYPE_PLANE Consts3D.THREED_SHAPE_TYPE_LINE Consts3D.THREED_SHAPE_TYPE_BILLBOARD Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL</p> <p>It is used to describe which kind of primitive is used to represent the bubble of the specified element. As usual, THREED_SHAPE_TYPE_BILLBOARD is the simplest choice. It works when ALARM_RENDER_ENABLE and ALARM_RENDER_ENABLE are set as true.</p>	
Style3D.ALARMATTACHMENT_MAPPINGTYPE	<p>Values are listed as below:</p> <p>Consts3D.MAPPINGTYPE_COLOR Consts3D.MAPPINGTYPE_COMMON</p>	<p>TWaver 3D for Flex uses primitives to represent the element's bubble. For more details please refer to Style3D.MAPPINGTYPE. It works when ALARM_RENDER_ENABLE and ALARM_RENDER_ENABLE are set to true..</p>
Style3D.ALARMATTACHMENT_MAPPINGCOMMON_PATH	<p>Style3D.MAPPING_COMMON_PATH</p>	<p>TWaver 3D for Flex uses primitives to represent the bubble of an element. For more details refer to Style3D.MAPPINGTYPE. It works when ALARM_RENDER_ENABLE and ALARM_RENDER_ENABLE are set as true.</p>
Style3D.ALARMATTACHMENT_LOCATION	Vector3D	<p>It defines the postion of the alarm bubble relative to the specified element. It should use the relative coordinate and it works when ALARM_RENDER_ENABLE and ALARM_RENDER_ENABLE are set as true.</p>
Style3D.ALARMATTACHMENT_SIZE	Vector3D	<p>It is the same as Style3D.PROPERTY_SIZE. It works only when ALARM_RENDER_ENABLE is set as true.</p>
Style3D.ALARMATTACHMENT_MATERIAL_COLOR	uint COLOR	<p>It is the same as Style3D.MATERIAL_COLOR. It works when ALARM_RENDER_ENABLE and ALARM_RENDER_ENABLE are set as true.</p>

 Note: For more details of the 3D styles, you can refer to the demos of TWaver 3D for Flex.

In TWaver 3D flex, the Element objects are displayed as predefined primitive models, such as a cube, a cone and a plane.

Primitive Objects

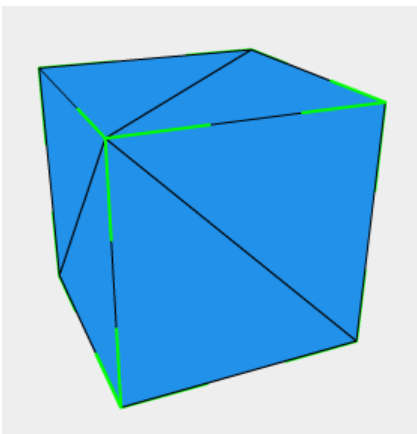
Each 3D object displayed is actually a collection of a number of base elements that have been combined to create the models you can see in a 3D scene. In order to reduce the complexity of creating those models, TWaver 3D has provided some primitive models. Developers can choose one or several to make up a 3D model. Creating a new instance of a primitive is very easy. What developers need to do is to create a new element instance and to assign an appropriate value of `Style3D.THREED_SHAPE_TYPE` style to that element. All the values are listed as below:

```
THREED_SHAPE_TYPE_CUBE
THREED_SHAPE_TYPE_SPHERE
THREED_SHAPE_TYPE_CONE
THREED_SHAPE_TYPE_PLANE
THREED_SHAPE_TYPE_LINE
THREED_SHAPE_TYPE_BILLBOARD
THREED_SHAPE_TYPE_EXTERNALMODEL
THREED_SHAPE_TYPE_BOXSCENE
THREED_SHAPE_TYPE_LATHEEXTRUSION
THREED_SHAPE_TYPE_LINEAREXTRUSION
```

All the values are defined in `twaver.Consts3D` class.

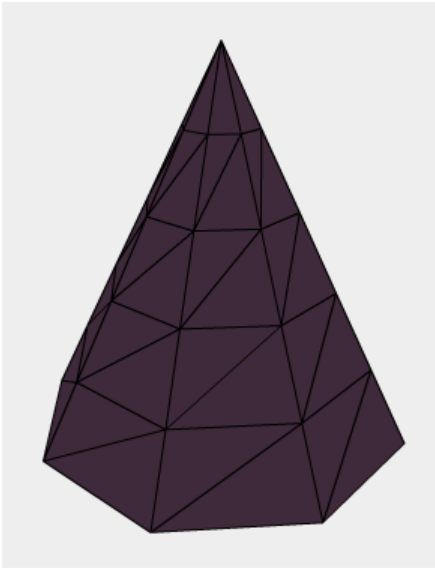
Cube

```
var cube:Node = new Node();
cube.setStyle(Style3D. THREED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_CUBE);
```



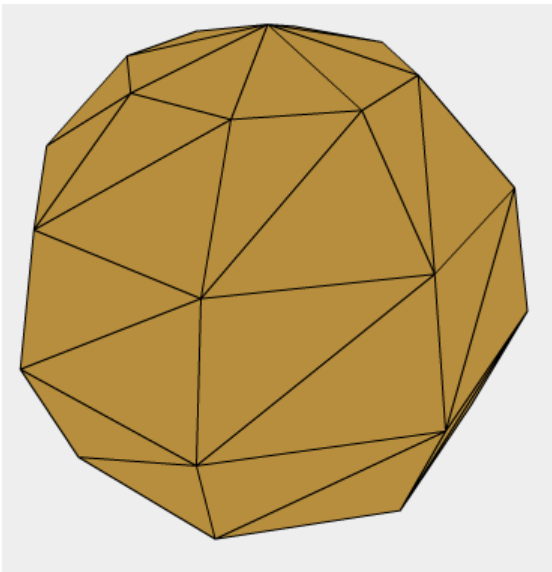
Cone

```
var cone:Node = new Node();
cone.setStyle(Style3D. THREED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_CONE);
```



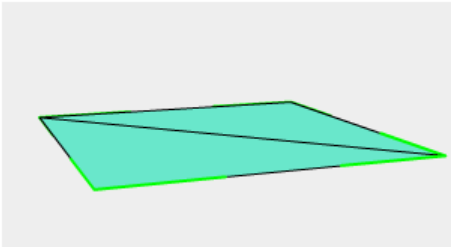
Sphere

```
var sphere:Node = new Node();
cube.setStyle(Style3D. THREEED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_SPHERE);
```



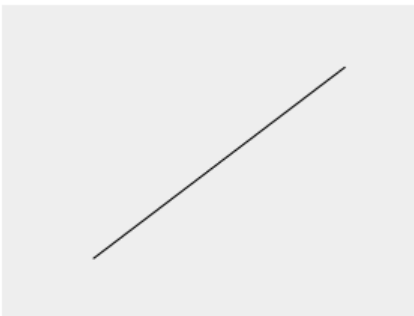
Plane

```
var plane:Node = new Node();
cube.setStyle(Style3D. THREEED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_PLANE);
```



Straight Line

```
var line:Node=new Node();
line.setStyle(Style3D.THREED_SHAPE_TYPE, Consts3D.THREED_SHAPE_TYPE_LINE);
line.setStyle(Style3D.LINE_SHAPE_POINTS, \[new Vector3D(0,0,0),new Vector3D(200,200,200)\]);
```



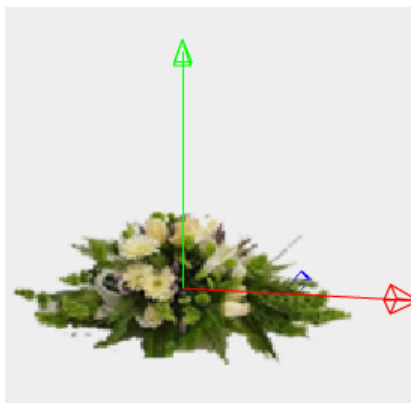
Scene Box

```
var box:Node=new Node();
box.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_BOXSCENE);
box.setStyle(Style3D.MAPPING_COMMON_PATH,"scenebox");
```




Billboard

```
var bill:Node=new Node();
bill.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_BILLBOARD);
bill.setStyle(Style3D.BILLBOARD_NORMAL_TEXTURE, "flower");
bill.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COMMON);
```



LinearExtrusion

[Embed(source="images/b.jpg")]

```

Utils.registerImageByClass("R",R);
public static const R:Class;
.....
var n:Node = new Node();
n.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_LINEAREXTRUSION);
var track:Array = [new Vector3D(0,0,-100),new Vector3D(300,0,-100),new Vector3D(300,0,100),new Vector3D(0,0,100)];
n.setStyle(Style3D.EXTRUSION_PROFILE,track);
n.setStyle(Style3D.EXTRUSION_THICKNESS,5);
n.setStyle(Style3D.EXTRUSION_OFFSET,100);
n.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_FRONT_COLOR,0x80EE80);
n.setStyle(Style3D.MAPPING_FRONT_TYPE,Consts3D.MAPPINGTYPE_COLOR);
n.setStyle(Style3D.MAPPING_TOP_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_TOP,"R");
n.setStyle(Style3D.MAPPING_BOTTOM_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_BOTTOM,"R");
n.setStyle(Style3D.MAPPING_BACK_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_BACK,"R");
n.setStyle(Style3D.MAPPING_LEFT_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_LEFT,"R");
n.setStyle(Style3D.MAPPING_RIGHT_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_RIGHT,"R");
n.setStyle(Style3D.EXTRUSION_RECENTER,true);

```



LatheExtrusion

```

var n:Node = new Node();
n.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_LATHEEXTRUSION);
var track:Array = [new Vector3D(100,0,0),new Vector3D(100,100,0)];
n.setStyle(Style3D.EXTRUSION_PROFILE,track);
n.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(0,90,0));
n.setStyle(Style3D.EXTRUSION_ROTATIONS,0.5);
n.setStyle(Style3D.EXTRUSION_THICKNESS,5);
n.setStyle(Style3D.EXTRUSION_DIVISIONS,24);
n.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_BACK_COLOR,0x80EE80);
n.setStyle(Style3D.MAPPING_BACK_TYPE,Consts3D.MAPPINGTYPE_COLOR);
n.setStyle(Style3D.MAPPING_TOP_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_TOP,"R");
n.setStyle(Style3D.MAPPING_BOTTOM_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_BOTTOM,"R");
n.setStyle(Style3D.MAPPING_FRONT_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_FRONT,"R");

```

```
n.setStyle(Style3D.MAPPING_LEFT_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_LEFT,"R");
n.setStyle(Style3D.MAPPING_RIGHT_TYPE,Consts3D.MAPPINGTYPE_COMMON);
n.setStyle(Style3D.MAPPING_CUBE_RIGHT,"R");
n.setStyle(Style3D.EXTRUSION_RECENTER,false);
```

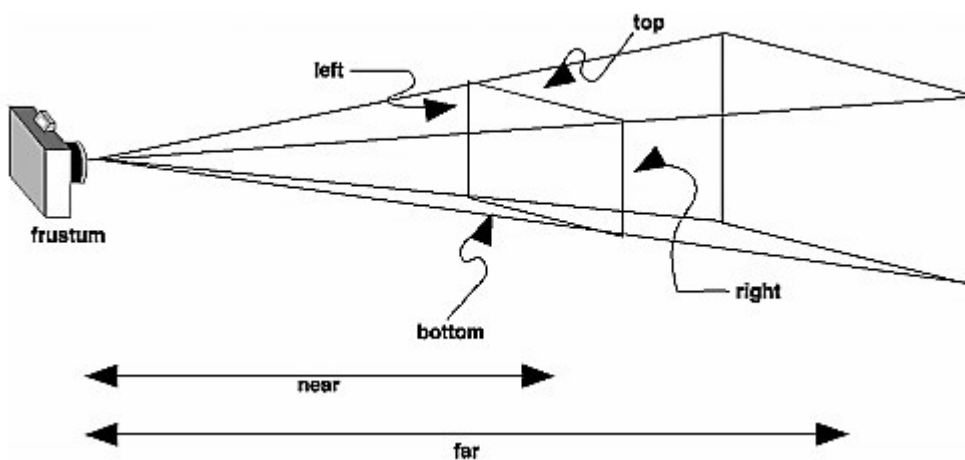


Camera

In TWaver 3D for Flex, a Camera object is used to simulate men's eye and serves as an observation site in the view. The Camera object can project 3D world scene to the 2D world that is necessary for representation on a computer screen. It influences what is visible in the view by calculating a projected image based on the position and the rotation of the camera, as the view is showing you the contents of the scene from your eyes.

In order to display a 3D scene on the screen, developers must apply some kind of camera on a Network3D component. TWaver 3D for Flex provides common camera and orbit(hover) camera. Developers can change the position of a camera by moving it to get different views.

```
network.cameraPosition = new Vector3D(100,10,-100);
```



The most unmistakable characteristic of perspective projection is foreshortening: the farther an object is from the camera, the smaller it will appear in the final image. This occurs because the viewing volume for a perspective projection is a frustum of a pyramid (a truncated pyramid whose top has been cut off by a plane parallel to its base). Objects that fall within the viewing volume are projected toward the apex of the pyramid, where the camera or viewpoint is. Developers can resize the frustum by assigning new values to the nearLimit or farLimit property of the Network3D object with the following code.

```
network.farLimit = 10000;
or
network.nearLimit = 0.01;
```

- [Common Camera](#)
- [Orbit or Hover Camera](#)

Common Camera

```
//new Network3D instance
var network:Network3D = new Network3D();
//initialize 3D scene by fill 3D models into the ElementBox
var box:ElementBox = network.elementBox;
var node:Node = new Node();
node.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_SPHERE);
node.setStyle(Style3D.PROPERTY_SIZE,new Vector3D(50,50,50));
node.setStyle(Style3D.PROPERTY_SPACE_LOCATION,new Vector3D(0,0,0));
node.setStyle(Style3D.PROPERTY_SMOOTH_LEVEL,Consts3D.SMOOTH_LEVEL_GREAT);
box.add(node);
//apply the camera on the network to display the 3D scene
var position:Vector3D = new Vector3D(0,0,-100);
var panAngle:Number = -180;
var tiltAngle:Number = 0;
network.applyCommonCamera(position,panAngle,tiltAngle);
```

When the component is displayed on the computer screen, users can rotate the view by dragging the mouse, moving forward/backward, and panning left/right with the direction keys.

With the common camera, developers can get view of the 3D scene in the first person.

Orbit or Hover Camera

If you want to navigate all sides of a 3D scene by using mouse, an ideal choice is to apply hover camera on the Network3D.

With the help of hover camera, users can pan, lift and move the camera around the 3D scene by dragging the mouse on the Network3D object. The distance between the camera and the center of the 3D scene determines the FOV.

Apply hover camera:

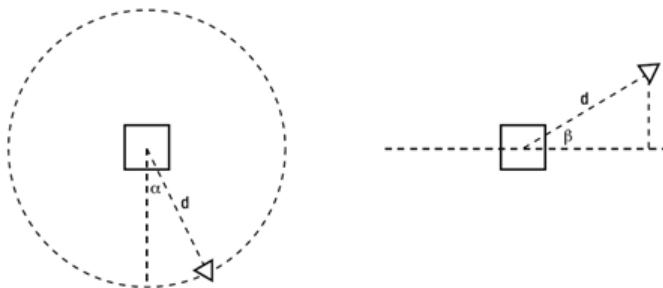
```
network.applyHoverCamera(panAngle,tiltAngle,distance);
```

panAngle: Stands for the pan angle of the hover camera. Default: -180 degree.

tiltAngle: Stands for the tilt angle of the hover camera. Default: 10 degree.

distance: Stands for the distance between the camera and the pivot of the scene. Default: 500.

The increment and decrement in the tiltAngle value causes the camera to increase or decrease its elevation angle as presented in the following image. The "distance" property defines the radius on which both rotations are performed. Two angles and a radius which are used to define a position in space in this manner are sometimes referred to as polar coordinates.

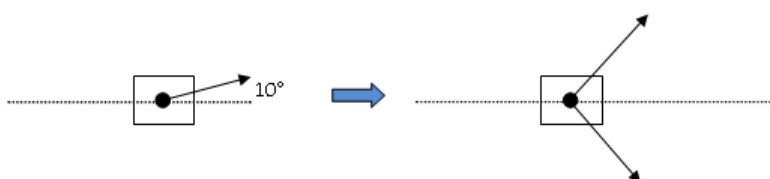


Note: The above is the schematic view of a simple scene in which the hover camera is used. The left figure shows the scene with α denoting the "panAngle" property and the right figure shows the scene displayed from a side view with β denoting the "tiltAngle" property of the camera. In addition, d denotes the "distance" property in both cases.

The hover camera's tilt angle is limited at 10° as default. The elevated view of the scene is locked. If you want to change the default settings and elevate the view as you want, you can reset the related properties of the Network3D object.

```
network.tiltAngleUpLimit = 60;  
network.tiltAngleLowLimit = -60;
```

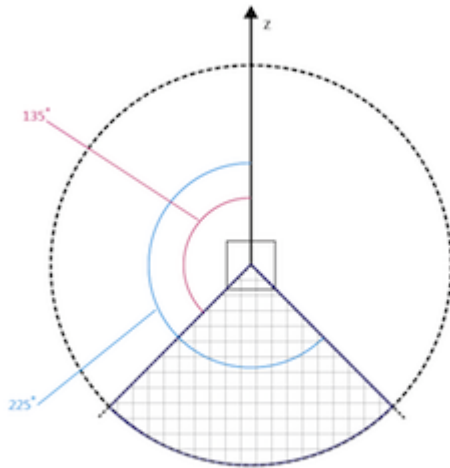
With the above segment, developers can unlock the elevated view of the 3D scene.



Users will be able to tilt the camera between -60° and 60° .

Developers can set the range among which users are able to pan the camera.

```
network.panAngleUpLimit = 225;
network.panAngleLowLimit = 135;
```



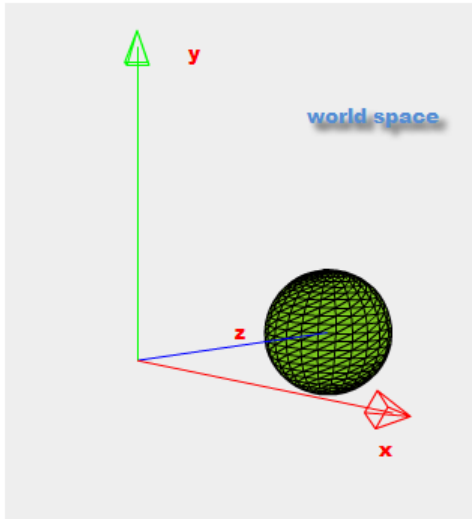
After applying the hover camera to the Network3D component, users can orbit around the 3D scene by dragging the mouse or zooming in/out the scene with the mouse wheel.

Coordinate System

Coordinates are used to position a 3D object in the scene. In TWaver 3D for Flex, the coordinates can be described from three references: global, parent and local coordinate.

World Space

The global coordinate system represents points or vectors relative to the origin of the scene. This coordinate system is also called world space. In TWaver 3D for Flex, the origin is located in the center of the Network3D component.

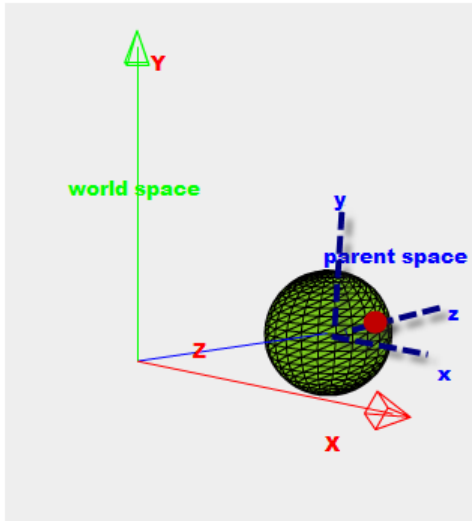


Every 3D object displayed in Network3D is positioned by global coordinate.

Note: If you want to move a 3D object in the scene, what you need to do is just to change its position in the world space.

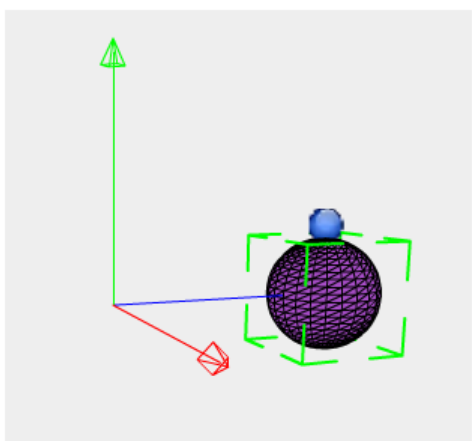
Parent Space

The parent coordinate system is used to represent points that are related to the position and direction of a 3D object's parent container. This coordinate system is also known as parent space.



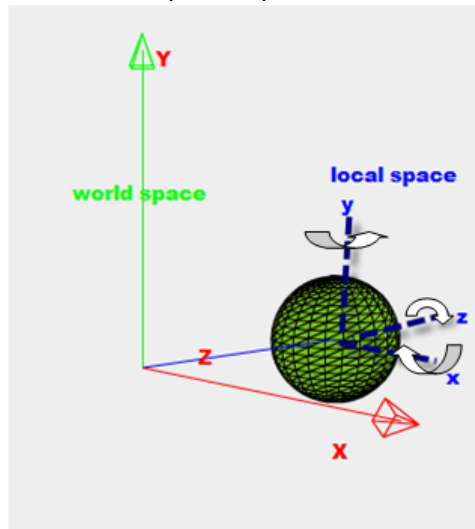
In TWaver 3D for Flex, the alarm attachment object is located in parent space.

```
var node:Node = new Node();
node.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_SPHERE);
node.setStyle(Style3D.PROPERTY_SIZE,new Vector3D(50,50,50));
node.setStyle(Style3D.PROPERTY_SPACE_LOCATION,new Vector3D(0,0,100));
node.setStyle(Style3D.PROPERTY_SMOOTH_LEVEL,Consts3D.SMOOTH_LEVEL_GREAT);
node.setStyle(Style3D.ALARM_BUBBLE_ENABLE,true);
node.setStyle(Style3D.ALARMATTACHMENT_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_BILLBOARD);
node.setStyle(Style3D.ALARMATTACHMENT_MAPPING_COMMON_PATH,"balloon");
node.setStyle(Style3D.ALARMATTACHMENT_MAPPINGTYPE,Consts3D.MAPPINGTYPE_COMMON);
//locate the alarm attachment in parent space
node.setStyle(Style3D.ALARMATTACHMENT_LOCATION,new Vector3D(0,30,0));
box.add(node);
var alarm:Alarm = new Alarm(null,node.id,AlarmSeverity.MAJOR);
box.alarmBox.add(alarm);
```



Local Space

The local coordinate system is used to represent points related to the direction of a 3D object. This system is



also known as local space.

In TWaver 3D for Flex, the rotation and the scaling of a 3D object can be operated in a local space.

Position 3D Objects

Position a cone object at the point (0,100,0) in global space.

```
var node:Node = new Node();
node.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_SPHERE);
node.setStyle(Style3D.PROPERTY_SIZE,new Vector3D(50,50,50));
node.setStyle(Style3D.PROPERTY_SPACE_LOCATION,new Vector3D(0,0,100));
box.add(node);
```

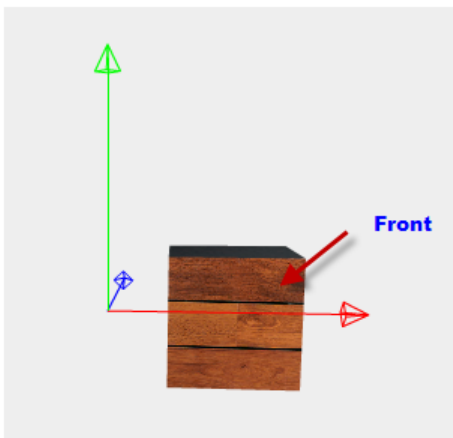
When moving the object in a 3D scene, developers should keep changing the Style3D.PROPERTY_SPACE_LOCATION values. For example:

```
var timer:Timer = new Timer(1000);
var z:Number = 10;
timer.addEventListener(TimerEvent.TIMER,function(evt:Event):void{
    target.setStyle(Style3D.PROPERTY_SPACE_LOCATION,new Vector3D(0,0,z++));
});
timer.start();
```

Rotate 3D Objects

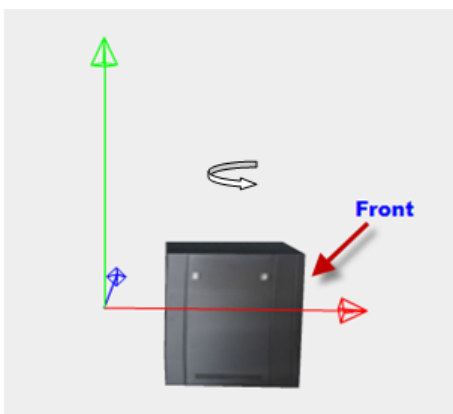
```
var node:Node = new Node();
node.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_CUBE);
node.name = name;
node.setStyle(Style3D.PROPERTY_SPACE_LOCATION,location);
node.setStyle(Style3D.PROPERTY_SIZE,size);
node.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COMMON);
node.setStyle(Style3D.MAPPING_CUBE_FRONT,"floorimage");
node.setStyle(Style3D.MAPPING_CUBE_LEFT,"CSide");
node.setStyle(Style3D.MAPPING_CUBE_RIGHT,"CSide");
node.setStyle(Style3D.MAPPING_CUBE_BACK,"CBack");
node.setStyle(Style3D.MAPPING_CUBE_TOP,"CTop");
node.setStyle(Style3D.MAPPING_CUBE_BOTTOM,"CTop");
box.add(node);
```

With the codes above, we can get a cube as shown in the following shot.



Now we rotate a cube object -90 degrees around y-axis in its local space.

```
node.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(0,-90,0));
```



Scale 3D Objects

Scale a sphere object uniformly along all three local axes.

```
// or along the local x, y or z axis individually.  
node.setStyle(Style3D.PROPERTY_SCALE,3);
```

We will get a magnified cube as shown in the following snap shot.



Materials

All the primitives defined by TWaver 3D for flex are built from geometric elements, such as triangles and line segments. In order that the renderer is able to draw these elements to the view, an definition on appearance should be required. In this TWaver 3D, this definition is referred to as a material and is described by some styles. In general, materials can be used to paint the surfaces of 3D objects with solid colors or bitmap images.

- [Bitmap Materials](#)
- [Color Material](#)
- [Alpha Effect](#)

Bitmap Materials

Bitmap materials are used to display a texture map, usually a jpg, png, or gif image file on the surface of a 3D object. TWaver 3D for Flex uses UV coordinates to manage the alignment of the texture map on the surface of a 3D object.

If you want to map an image onto a face of the 3D object, you can use the following segment:

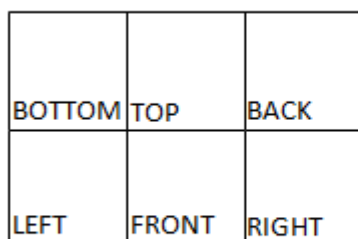
```
var sphere:Node = new Node();
sphere.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_SPHERE);
sphere.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COMMON);
sphere.setStyle(Style3D.MAPPING_COMMON_PATH,"floorimage");
```

In TWaver 3D for Flex, you can apply an image onto the surface of a Cube, a Sphere, a Plane, a Cone, a Cylinder and a Billboard object. When applying an image on a Cube object with the codes above, the entire image will be mapped on each face.

If you want to assign different materials for each face of a Cube object, you should use the code as below:

```
var cube:Node = new Node();
cube.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_CUBE);
cube.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_MAP6);
cube.setStyle(Style3D.MAPPING_COMMON_PATH,"PC");
```

As we know, a cube object has six surfaces. To describe the 6 surfaces through only one image, a developer should make a texture image that should be split up into two rows and three columns, with each of the six divisions being applied to one of the surface of the cube. An example of such a texture is shown in the following figures:



The rule



The texture image

We can get a result with the texture above:



front/left/top



right/back/bottom

You also can use the following codes to apply the 6 texture files onto the 6 different surfaces of a ComplexCube object to get the same result as above:


```
var node:Node = new Node();
node.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_COMPLEXCUBE);
node.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COMMON);
node.setStyle(Style3D.MAPPING_CUBE_FRONT,"CFront");
node.setStyle(Style3D.MAPPING_CUBE_LEFT,"CLeft");
node.setStyle(Style3D.MAPPING_CUBE_RIGHT,"CRight");
node.setStyle(Style3D.MAPPING_CUBE_BACK,"CBack");
node.setStyle(Style3D.MAPPING_CUBE_TOP,"CTop");
node.setStyle(Style3D.MAPPING_CUBE_BOTTOM,"CBottom");
```

The resources referred to are listed in the following:



left.png



front.png



right.png



back.png



top.png



bottom.png

Note:The texture files which are used above have been embedded into the code as resources and have been registered into TWaver with the following codes.

```
[Embed(source="/test/resource/images/floor.jpg")]
private static var FloorImage:Class;
[Embed(source="/test/resource/images/b128.png")]
private static var PCImage:Class;
[Embed(source="/test/resource/images/front.png")]
private var CFront:Class;
[Embed(source="/test/resource/images/back.png")]
private var CBack:Class;
[Embed(source="/test/resource/images/left.png")]
private var CLeft:Class;
[Embed(source="/test/resource/images/right.png")]
private var CRight:Class;
[Embed(source="/test/resource/images/top.png")]
private var CTop:Class;
[Embed(source="/test/resource/images/bottom.png")]
private var CBottom:Class;
...
Utils.registerImageByClass("floorimage",FloorImage);
Utils.registerImageByClass("PC",PCImage);
Utils.registerImageByClass("CFront",CFront);
Utils.registerImageByClass("CBack",CBack);
Utils.registerImageByClass("CSide",CLeft);
Utils.registerImageByClass("CRight",CRight);
```

```
Utils.registerImageByClass("CTop",CTop);
Utils.registerImageByClass("CBottom",CBottom);
```

Developers can not only use the embedded resource as the texture, but also specify the URL of a texture. TWaver 3D will dynamically load the remote image source with a loader. The following codes show you how to use the URL.

```
cube.setStyle(Style3D.MAPPING_COMMON_PATH,"http://www.servasoftware.com/resource/test.jpg");
```

Developers can directly use a BitmapData object as the source of a texture.

```
[Embed(source="/test/resource/images/top.png")]
private var CTop:Class;
.....
cube.setStyle(Style3D.MAPPING_COMMON_PATH,new CTop().bitmapData);
```

In addition to the primitives above, TWaver 3D also supports the importing of external model files to build a 3D scene. The following formats can be supported: 3ds, md2, obj, and dae. When designing these kinds of files, designers should correctly set up the uv coordinates in the files.



The 3d file stores the filenames of the textures in ASCII code, so the filenames should not include non-ASCII code.

When exporting a 3ds file, some tool, such as 3DMax, will store the filenames with uppercase letters, but the real filenames are lowercase letters. In order to load the correct texture files when loading a 3ds file, developers should use the following codes before loading them.

```
Manager3D.registerGlobalSetting(Style3D.ASSETS_LOWERCASE,true);
```

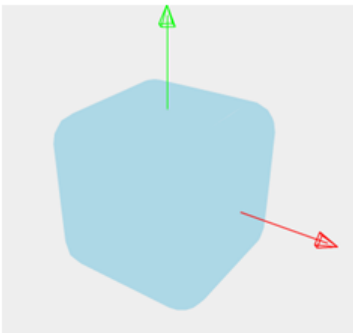
Color Material

TWaver 3D for Flex provides color renderer function to display a 3D object with the colored surface when appending the specified color.

Only Color

```
var node:Node = new Node();  
node.setStyle(Style3D.THREED_SHAPE_TYPE,  
Consts3D.THREED_SHAPE_TYPE_ROUNDEDCUBE);  
node.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COLOR);  
node.setStyle(Style3D.MATERIAL_COLOR,0xADD8E6);
```

You will get a rounded cube which is filled by the specified color(value:0xADD8E6).



Append Color

If a 3D object has been rendered with image material, TWaver 3D for Flex can help you append the specified color to the material.

```
var node:Node = new Node();
node.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_MAP6);
node.setStyle(Style3D.PROPERTY_SIZE,size);
node.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_CUBE);
node.setStyle(Style3D.MAPPING_COMMON_PATH,"PC");
node.setStyle(Style3D.MATERIAL_COLOR,0xA055FF);
```

The following snap shows the result:



If an alarm occurs in an element, TWaver 3D for Flex will append the color of the highest alarm severity to the 3D shape of that element as default. For more details please refer to Render Alarms.

Alpha Effect

To control the transparency of the surface of a 3D object, developers can assign alpha values to a 3D object.

```
var leftWall:Node = new Node();
leftWall.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_ROUNDEDCUBE);
leftWall.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(0,90,0));
leftWall.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_COLOR);
leftWall.setStyle(Style3D.MATERIAL_COLOR,0xADD8E6);
//set the alpha value as 0.9
leftWall.setStyle(Style3D.MATERIAL_ALPHA,0.9);
```



If the 3D object has been rendered with image material, then its surfaces can be covered with a transparent layer.

```
var node:Node = new Node();
node.setStyle(Style3D.MAPPINGTYPE,Consts3D.MAPPINGTYPE_MAP6);
node.setStyle(Style3D.PROPERTY_SIZE,size);
node.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_CUBE);
node.setStyle(Style3D.MAPPING_COMMON_PATH,"PC");
node.setStyle(Style3D.MATERIAL_ALPHA,0.2);
```



Model File

As we have discussed, developers can create Element objects manually in codes to build a 3D scene, but it will take too much time to build a complex 3D scene. TWaver 3D supports importing model files to re-build a pre-designed scene.

- [Import External Models](#)
- [TWaver Format Model File](#)

Import External Models

UTWaver 3D for Flex can load some kinds of 3D model files to help build a 3D scene. TWaver 3D supports 3ds, obj and md2 files.

- [Load DAE models](#)
- [Load 3DS models](#)
- [Load Obj models](#)
- [Load md2 models](#)

Load DAE models

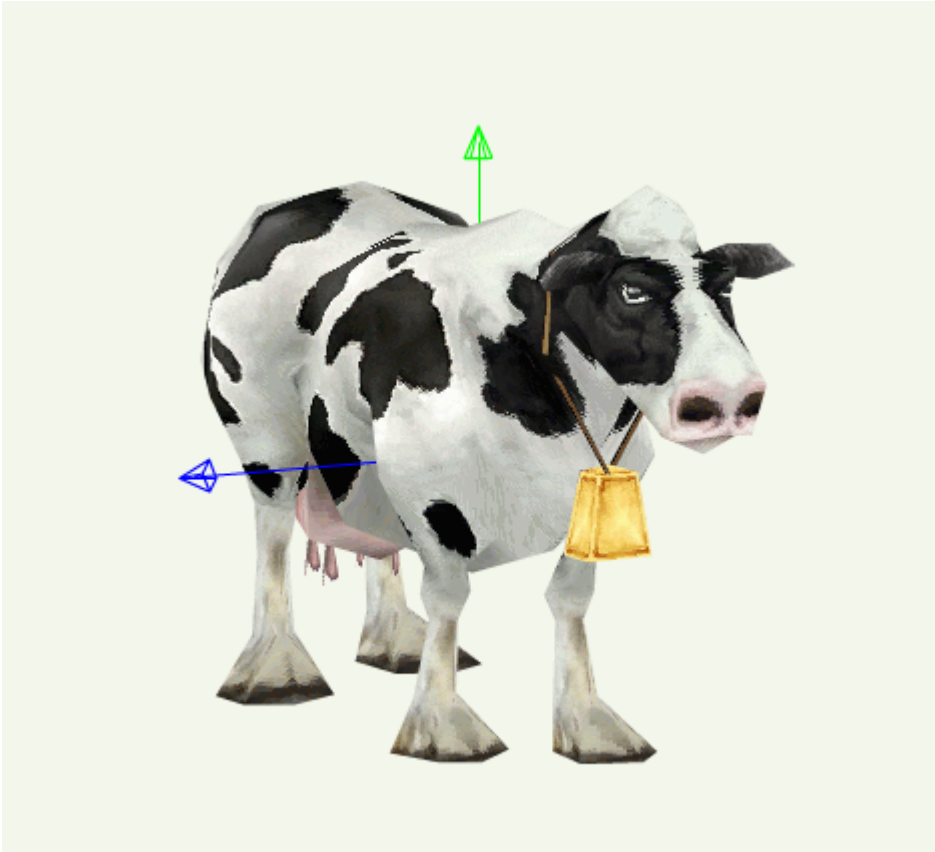
COLLADA is a COLLABorative Design Activity for establishing an interchange file format for interactive 3D applications. COLLADA is managed by the nonprofit technology consortium, the Khronos Group. COLLADA defines an open standard XML schema for exchanging digital assets among various graphics software applications that might otherwise store their assets in incompatible file formats. COLLADA documents which describe digital assets are XML files, usually identified with a .dae (digital asset exchange) filename extension. For more details of COLLADA, please refer to <http://en.wikipedia.org/wiki/COLLADA>. .dae file will be used in every kind of terminal equipment and OS platform. With TWaver 3D in hand, developers can easily use this kind of model file.

```
[Embed(source="/models/cow.dae",mimeType="application/octet-stream")]
public static const Cow:Class;
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(0,180,0));
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_DAE);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,Cow);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,1000);
box.add(modelNode);
```

Developers can access the file through the url.

```
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(0,180,0));
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_DAE);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,"http://www.server.com/models/cow.dae");
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,1000);
box.add(modelNode);
```

The following snapshot is the result loaded by the codes above.



Load 3DS models

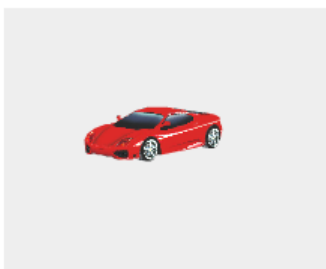
TWaver 3D for Flex supports 3ds format files.

```
[Embed(source="/models/ferrari.3ds",mimeType="application/octet-stream")]
public static const Ferrari:Class;
//load embedded 3ds file
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(90,180,0));
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_3DS);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,Ferrari);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,0.5);
box.add(modelNode);
```

To reduce the size of the deployment package, developers can put the external model files and material files at the server side and access the resources through http protocol.

```
//load 3ds file from server side
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(90,180,0));
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_3DS);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,
"http://www.server.com/res/ferrari.3ds");
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,0.5);
box.add(modelNode);
```

Developers can get the following result with the code segments above.



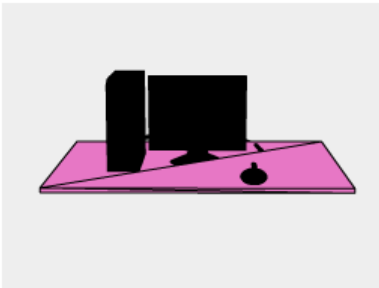
Load Obj models

Load external obj format files

```
[Embed(source="/models/pc.obj",mimeType="application/octet-stream")]
public static const PC:Class;
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_OBJ);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,PC);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,0.1);
box.add(modelNode);
```

DeveloperS can get the file from the server side with the following code:

```
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_OBJ);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,
"http://www.server.com/res /pc.obj");
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,0.1);
box.add(modelNode);
```



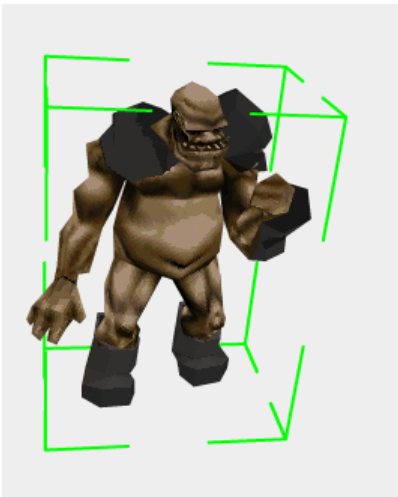
Load md2 models

```
[Embed(source="/models/ogro.md2",mimeType="application/octet-stream")]
public static const MD:Class;
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(90,0,0));
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_MD2);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,MD);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,0.05);
box.add(modelNode);
```

Developers can access the file from the server side

```
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.PROPERTY_ROT_ANGLE,new Vector3D(90,0,0));
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_MD2);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,
"http://www.serva.com/res/ogro.md2");
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SCALE,0.05);
box.add(modelNode);
```

After loading the md2 file mentioned above, you can get a result shown in this snap.



Note: As for the management of the external model resources, you can refer to Resource Management.

TWaver Format Model File

TWaver 3D for Flex provides a visual design application named TWaver 3D Editor. The tool can help users build a room scene or an equipment and is able to save the scene model into an XML which can be parsed by TWaver products.

Resource Management

As is discussed above, image material relies on texture files and external 3d models needs related model files. All these two kinds of files are called resources in TWaver 3D for Flex. Developers can embed the resources into their projects or dynamically load them at runtime by http protocol.

Embed Resources into Project

Texture files can be embedded into the project.

```
[Embed(source="/ resource/images/floor.jpg")]
private static var FloorImage:Class;
.....
//register the resource into TWaver
Utils.registerImageByClass("floorimage",FloorImage);
....
//use the embedded resource
node.setStyle(Style3D.MAPPING_COMMON_PATH,"floorimage");
```

Model files can be embedded into the project.

```
[Embed(source="/test/resource/models/f360.3ds", mimeType="application/octet-stream")]
private var Ferrari:Class;
.....
//load the embedded models
var modelNode = new Node();
modelNode.setStyle(Style3D.THREED_SHAPE_TYPE,
Consts3D.THREED_SHAPE_TYPE_EXTERNALMODEL);
modelNode.setStyle(Style3D.EXTERNAL_MODEL_TYPE,
Consts3D.EXTERNAL_MODEL_TYPE_3DS);
modelNode.setStyle(Style3D. EXTERNAL_MODEL_SOURCE,Ferrari);
```


Access Resources at Runtime

TWaver supports the access to resources at runtime. When using the external resource in TWaver 3D for Flex, developers can assign the full path of the resource to the 3D element.

Access the texture files through http protocol.

```
node.setStyle(Style3D.MAPPING_COMMON_PATH, "http://www.servasoft.com/res/r.png");
```

Access the external model files through http protocol.

```
modelNode.setStyle(Style3D.EXTERNAL_MODEL_SOURCE,  
"http://www.servasoft.com/res/Ferrari.3ds");
```

Note: In an application, if developers want to reduce the size of the deployment package, it will be a good choice to use full URL paths to access external resources.

Interactions

Developers can add more interactions between the 3D scene and users. In general, users always use mouse or keyboard as a trigger to interact with the system interface, such as clicking a specified element, or pressing some keys.

- [Use camera](#)
- [Mouse Interaction](#)
- [Use other Events](#)

Use camera

In order to control the camera in a 3D scene, developers must have the ability to get and set the state of the camera. TWaver 3D provides simple methods to do that.


1. Retrive current camera's state

Developers can get current camera's zoom, position, pan angle and tilt angle.

```
//Gets current camera's zoom;
var zoom:Number = network.zoom;
//Gets current camera's position;
var position:Vector3D = network.cameraPosition;
```

If a hover camera is used in the scene, developers can get the pan angle, tilt angle, and the distance between the camera and the pivot of the scene.

```
//Gets pan angle;
var panAngle:Number = network.panAngle;
//Gets tilt angle;
var tiltAngle:Number = network.tiltAngle;
//Gets distance;
var distance:Number = network.hoverViewDistance;
```

 Please refer to [Event-driven](#)

2. Move the camera in the scene

When the common camera is used in the scene, developers can move the the position of the camera to simulate walking in the scene.

```
//when using common camera
network.cameraPosition = new Vector3D(1000,200,0);
```

or keep invoking applyHoverCamera/applyCommonCamera to change the camera's position.

3. Control the pan/tilt angle.

Developer are also able to control the pan angle and tilt angle with the following code.

```
//pan angle
network.panAngle = 30;
//tilt angle
network.tiltAngle = 20;
```

Developers can also set the range among which the hover camera can be panned or tilted. Please refer to [Orbit or Hover Camera](#)

Mouse Interaction

You can use the method `addMouseOnElementListener/addMouseOnCanvasListener` to add a mouse listener.

1. DoubleClick Element

```
network3d.addMouseOnElementListener(Mouse3DEvent.MOUSE_DOUBLECLICK_ON_ELEMENT,
function(e:Mouse3DEvent):void{
    var element:IElement=e.source as Element;
    if (element){
        trace("double click " + element.id);
    }
});
```

2. Click an Element

```
network3d.addMouseOnElementListener(Mouse3DEvent.MOUSE_DOUBLECLICK_ON_ELEMENT,
function(e:Mouse3DEvent):void{
    var element:IElement=e.source as Element;
    if (element){
        trace("double click " + element.id);
    }
});
```

3. DoubleClick Canvas

```
network3d.addMouseOnCanvasListener(Mouse3DEvent.MOUSE_DOUBLECLICK_ON_CANVAS,
function(e:Mouse3DEvent):void{
    trace("MOUSE_DOUBLECLICK_ON_CANVAS");
});
```

4. Click an Canvas

```
network3d.addMouseOnCanvasListener(Mouse3DEvent.MOUSE_CLICK_ON_CANVAS,
function(e:Mouse3DEvent):void{
    trace("MOUSE_CLICK_ON_CANVAS");
});
```

Use other Events

TWaver 3D provides several other kinds of Events to help developers monitor the states of the 3D scene, such as loading external model files and changing the camera's position, etc.

- [Use Camera3DEvent](#)
- [Use ModelLoadEvent](#)

Use Camera3DEvent

A Camera3DEvent object will be dispatched when the camera state of a scene is changed. Moving camera, and resetting the lens property of a camera will generate a Camera3DEvent object. Developers can use this kind of Event with the following code:

```
network.addEventListener(Camera3DEvent.CAMERA_UPDATE,onCameraUpdated);  
//  
private function onCameraUpdated(evt:Camera3DEvent):void{  
    var pos:Vector3D = network.cameraPosition;  
    trace("current camera position is "+pos);  
}
```

Use ModelLoadEvent

ModelLoadEvent objects will be dispatched when loading external model files and the progress of loading models can be monitored with the ModelLoadEvent objects. They are only dispatched by Util3D. Developers can handle this kind of Event with the following code :

```
Util3D.addEventListener(ModelLoadEvent.LOAD_SUCCESS,onModelLoaded);
//
function onModelLoaded(evt:ModelLoadEvent):void{
    trace(evt.elementId + " finished loading the external model");
}

//
Util3D.addEventListener(ModelLoadEvent.LOAD_PROGRESS,onLoadingModel);
//
function onLoadingModel(evt:ModelLoadEvent):void{
    trace(evt.elementId + " is loading the external model");
}

//
Util3D.addEventListener(ModelLoadEvent.LOAD_ERROR,onLoadingError);
//
function onLoadingError(evt:ModelLoadEvent):void{
    trace("Error occurs when " + evt.elementId + " is loading the external model, and error message is "+evt.cause);
}
```

Effect Functions On Network3D

- [SelectedEffectFunction](#)
- [SelectFunction](#)
- [SelectedColorFunction](#)
- [SelectedWidthFunction](#)
- [SelectedGlowFunction](#)
- [SelectedBlurFunction](#)
- [SelectedScaleFunction](#)

SelectedEffectFunction

This function sets the display effect of selected objects and provides the following selected results:

Consts3D.EFFECT_TYPE_SCALE

Consts3D.EFFECT_TYPE_FRAMEWORK

The default implementation :

```
public static function DEFAULT_SELECT_EFFECT_FUNCTION(element:Element):String{  
    var type:* = element.getStyle(Style3D.SELECTED_EFFECT_TYPE);  
    if(!type){  
        type = Consts3D.EFFECT_TYPE_FRAMEWORK;  
    }  
    return type;  
}
```

SelectFunction

You can use the return value of this function to determine whether an element can be selected or not.

The default implementation :

```
public static function DEFAULT_SELECTION_FUNCTION(element:Element):Boolean{  
    return true;  
}
```

SelectedColorFunction

If you select the effect of `Consts3D.EFFECT_TYPE_FRAMEWORK`, this function will return the Number value of the selected network element to control the border width.

The default implementation :

```
public static function DEFAULT_SELECT_WIDTH_FUNCTION(element:Element):Number{  
    return element.getStyle(Style3D.SELECTED_FRAMEWORK_WIDTH);  
}
```

SelectedWidthFunction

If you select the effect of `Consts3D.EFFECT_TYPE_FRAMEWORK`, this function will return the Number value of the selected network element to control the border width.

The default implementation :

```
public static function DEFAULT_SELECT_WIDTH_FUNCTION(element:Element):Number{  
    return element.getStyle(Style3D.SELECTED_FRAMEWORK_WIDTH);  
}
```

SelectedGlowFunction

If you select the effect of `Consts3D.EFFECT_TYPE_GLOW`, this function will return the `GlowFilter` filter to control the display.

The default implementation :

```
public static function DEFAULT_SELECT_GLOW_FUNCTION(element:Element):GlowFilter{
    var filter:GlowFilter = element.getStyle(Style3D.SELECTED_EFFECT_FILTER) as GlowFilter;
    if(!filter){
        filter = new GlowFilter();
    }
    return filter;
}
```

SelectedBlurFunction

If you select the effect of `Consts3D.EFFECT_TYPE_BLUR`, this function will return the `BlurFilter` filter to control the display.

The default implementation :

```
public static function DEFAULT_SELECT_BLUR_FUNCTION(element:Element):BlurFilter{
    var filter:BlurFilter = element.getStyle(Style3D.SELECTED_EFFECT_FILTER) as BlurFilter;
    if(!filter){
        filter = new BlurFilter();
    }
    return filter;
}
```

SelectedScaleFunction

If you select the effect of `Consts3D.EFFECT_TYPE_SCALE`, this function will return a `Number` value to control the display zoom ratio.

The default implementation :

```
public static function DEFAULT_SELECT_SCALE_FUNCTION(element:Element):Number{  
    return 1.1;  
}
```

Render Alarms

AlarmEnableFunction

This function returns a true / false value to control whether to display the alarm or not.

The default implementation :

```
public static function DEFAULT_ALARM_RENDER_ENABLE_FUNCTION(element:Element):Boolean{
    var v:* = element.getStyle(Style3D.ALARM_RENDER_ENABLE);
    if(v is Boolean){
        return(v as Boolean);
    }
    return true;
}
```

AlarmColorFunction

This function returns the value of the color which is used to render the element on which some alarm occurs.

The default implementation :

```
public static function DEFAULT_ALARM_COLOR_FUNCTION(element:Element):Number{
    if(element.alarmState.highestNewAlarmSeverity){
        return element.alarmState.highestNewAlarmSeverity.color;
    }
    if(element.alarmState.propagateSeverity){
        return element.alarmState.propagateSeverity.color;
    }
    return NaN;
}
```