

Tutorial

Reference Manual

3D API

WireFusion 4.1



Contents

INTRODUCTION.....	1
About this Manual	1
REQUIREMENTS	2
User Requirements	2
System Requirements.....	2
DEFINITIONS	3
WireFusion W3F Format.....	3
X3D and VRML Definitions	3
WireFusion 3D API Definitions	4
BASICS	6
Get the running scene.....	8
Get a named node	8
Get a field.....	8
Read and change a field	9
Load new textures.....	9
Load an external 3D file	9
Add and remove a 3D file	9
Replace a 3D file.....	10
EXCERCISES	11
Exercise: Adding a 3D file to a 3D scene	11
Exercise: Changing the texture of an object.....	18
APPENDIX: SUPPORTED NODES AND FIELDS	21
Appearance.....	21
DirectionalLight	21
Group	22
Material	23
PointLight	24
Shape.....	25
Texture	25
TextureTransform	25
TimeSensor.....	26
TouchSensor.....	26
Transform.....	27
Viewpoint.....	28
INDEX	29

Introduction

The WireFusion 3D API (Application Program Interface) allows advanced users to reach and control parameters dynamically in a 3D scene while running a WireFusion 3D presentation.

The programming and connection to the 3D scene is done with the built-in WireFusion Java object. Selected nodes and fields can be dynamically changed, allowing the control of material properties, objects, cameras, lights, textures, positions, rotations, animations etc. It is also possible to dynamically add and remove 3D models and textures to running presentations.

Please send comments and feedback regarding this manual or the software to contact@demicron.com

About this Manual

In this manual we will introduce the WireFusion 3D API, explain the basics of how to control a 3D model and scene, and also go through some basic exercises.

Requirements

User Requirements

To take full advantage of the 3D API it is recommended that you know the Java programming language, or any other programming language. Non-programmers can also take advantage of the 3D API by re-using already made code or by copying and pasting code into the WireFusion Java object.

If you've never used WireFusion before, then it is highly recommended that you work through the tutorial **Getting Started, Volume I**, which requires no former WireFusion knowledge. You should also read the WireFusion **Java** manual, which explains how to write Java code in WireFusion using the Java object.

The full 3D API can be found on the Demicron web site (www.demicron.com/wirefusion/api)

System Requirements

In order to use the 3D API you have to have either the WireFusion Enterprise edition 4.1.12 (or higher) or the WireFusion Educational edition 4.1.12 (or higher).

Definitions

WireFusion W3F Format

In order to control your 3D scene dynamically using the 3D API you need to know some basic things about the X3D and VRML format, and about the W3F format (WireFusion 3D Format), which is the 3D format used internally in WireFusion. When an X3D or a VRML file is imported to WireFusion, then it is automatically converted to the WireFusion 3D Format (.w3f).

X3D and VRML Definitions

Some X3D and VRML definitions.

Scene Graph

The scene graph contains a hierarchy of **nodes**, which describe objects and their properties.

Nodes

Nodes are the fundamental component of a **scene graph** and are used to represent real-world objects. Each node in a scene graph is an instance of existing **node types**. Nodes contain **fields**.

Node Types

A characteristic of each **node** that describes, in general, its particular **field** semantics. Each **node type** has a fixed set of **fields**. For example, Shape, Material, DirectionalLight and TouchSensor are node types (see supported node types below).

Fields

A property or attribute of a **node**. **Fields** may contain various kinds of data and one or many values. Each field has a default value.

```

DEF Box01 Transform (
  translation -1.397 0.6373 0
  rotation -1 0 0 -1.571
  children [
    Shape (
      appearance DEF _1_ _Default Appearance (
        material Material (
          diffuseColor 0.5882 0.5882 0.5882
          ambientIntensity 1.0
          specularColor 0 0 0
          shininess 0.145
          transparency 0
        )
        texture ImageTexture (
          url "CARPTBLU.JPG"
        )
      )
      geometry DEF Box01_FACES(1) IndexedFaceSet (
        coord DEF Box01_COORD Coordinate { point [
          -0.9935 0 0.9841, 0.9935 0 0.9841, -0.9935 0 -0.9841,
          -0.9935 1.068 -0.9841, 0.9935 1.068 -0.9841 ] }
        coordIndex [
          0, 2, 3, -1, 3, 1, 0, -1, 4, 5, 7, -1, 7, 6, 4, -1, 0, 1, 5, -1
          2, 0, 4, -1, 4, 6, 2, -1 ]
      )
    ]
  ]
)
DEF Sphere01 Transform (
  translation 1.472 -0.02812 0
  rotation -1 0 0 -1.571
  children [
    Shape (
      appearance DEF _1_ _Default

```

-- Nodes

— Fields

Figure 1: Part of a VRML code, specifying the scene graph

WireFusion 3D API Definitions

The representation of the *scene graph* in the WireFusion 3D API is as follows.

Scene Graph

The X3D/VRML *scene graph* is represented in the 3D API with a hierarchy of [WF3DNode](#) classes. The [WF3DScene](#) class extends [WF3DNode](#) and acts as the root node for the scene graph.

Nodes

Every X3D/VRML *node* is represented in the 3D API as an instance of a [WF3DNode](#) class.

Node Types

All X3D/VRML *node types* are represented in the 3D API with the same [WF3DNode](#) class.

Fields

Every X3D/VRML **field** is represented in the 3D API as an instance of a [WF3DField](#) class.

NOTE: The full 3D API can be found at the Demicron site (www.demicron.com/wirefusion/api/).

Basics

You access the **scene graph** through a `WF3DScene` object, which can be retrieved by using the static methods provided in class `WF3DScene`. In the below examples we refer to the VRML code seen in Figure 2.

```

DEF Box01 Transform {
  translation -1.397 0.6373 0
  rotation -1 0 0 -1.571
  children [
    Shape {
      appearance DEF _1__Default Appearance {
        material Material {
          diffuseColor 0.5882 0.5882 0.5882
          ambientIntensity 1.0
          specularColor 0 0 0
          shininess 0.145
          transparency 0
        }
        texture ImageTexture {
          url "CARPTBLU.JPG"
        }
      }
      geometry DEF Box01_FACES(1) IndexedFaceSet {
        coord DEF Box01_COORD Coordinate { point [
          -0.9935 0 0.9841, 0.9935 0 0.9841, -0.9935 0 -0.9841,
          -0.9935 1.068 -0.9841, 0.9935 1.068 -0.9841 ] }
        coordIndex [
          0, 2, 3, -1, 3, 1, 0, -1, 4, 5, 7, -1, 7, 6, 4, -1, 0, 1, 5, -1
          2, 0, 4, -1, 4, 6, 2, -1 ]
      }
    ]
  }
}

DEF Sphere01 Transform {
  translation 1.472 -0.02812 0
  rotation -1 0 0 -1.571
  children [
    Shape {
      appearance DEF _1__Default
    }
  ]
}

```

-- Nodes
— Fields

Figure 2: VRML code

To retrieve the currently running scene containing the scene graph of for example the file "box.w3f", you call the static method `WF3DScene.getRunningScene()` as follows:

```
WF3DScene boxScene = WF3DScene.getRunningScene("box.w3f");
```

NOTE: Do not call the static `WF3DScene`-methods in the `init()`-method in the Java object, since the 3D scene may not have been initialized. Instead, call `enableSysEv(START)` and do all calls from the `handleSysEv(Event ev)` method.

Since `WF3DScene` extends `WF3DNode`, you can use the methods found in `WF3DNode` to access the nodes of the scene graph. To access any named node in the scene graph you use the `getNode(String)` method as follows:

```
WF3DNode boxTransformNode = boxScene.getNode("Box01");
```

You can now access the fields of a node by calling the `getField(int fieldIndex)` method. This method takes an integer as parameter, which indicates what field to retrieve. In the 3D API there is a package named **nodes** containing classes, one for each node type, where constants for all their fields are found. For instance, to access the field **translation** in a Transform node we would do:

```
WF3DField translationField =
    boxTransformNode.getField(nodes.Transform.translation);
```

In the `WF3DField` class there is a collection of `get` and `set` methods to be used to access the contained java type or object. Every `WF3DField` object has a type that specifies what X3D/VRML field type it represents, and this decides what functions you can call. For instance, by checking the Transform node in the 3D API we see that translation is of type `SFVec3f`. After checking the conversion table in class `WF3DField` we know this corresponds to a float array with three elements, so to retrieve the Java instance of the translation field we can use the `getFloatArray()` method as follows:

```
float[] translationFloatArray = translationField.getFloatArray();
```

We can now for example read the z-position by accessing its second element:

```
float z = translationFloatArray[2];
```

To set the field to a new value we can use the `set(Object newValue)` method, and as a parameter use a float array with three elements. We could for instance change the `translationFloatArray` and call `set()` with the `translationFloatArray` as parameter like this:

```
translationField.set(translationFloatArray);
```

NOTE: You need to be very careful when using the `set` and `get` methods for a field. Calling the wrong function on the field will result in an error.

Returning to the `boxTransformNode`, we can access the nodes in its children field by calling:

```
WF3DField childrenField =
    boxTransformNode.getField(nodes.Transform.translation);
WF3DNode[] childrenNodes = childrenField.getNodeArray();
```

The same thing can however be achieved more compactly by calling the `getChildren()` method of a `WF3DNode`:

```
WF3DNode[] childrenNodes = boxTransformNode.getChildren();
```

Checking the X3D/VRML-file we know that the first child node is a Shape node:

```
WF3DNode shapeNode = childrenNodes[0];
```

We can now access the Appearance node found in the appearance field by:

```
WF3DNode appearanceNode =  
    shapeNode.getField(nodes.Shape.appearance).getNode();
```

Or we could use the shorter `getNode(int fieldIndex)` method instead:

```
WF3DNode appearanceNode = shapeNode.getNode(nodes.Shape.appearance);
```

If we wanted to change the transparency of this Shape to 50 percent, we would continue to retrieve the material node and the transparency field:

```
WF3DNode materialNode =  
    appearanceNode.getNode(nodes.Appearance.material);  
WF3DField transparencyField =  
    materialNode.getField(nodes.Material.transparency);
```

Transparency is of type *SFFloat* which means we can use the `set(float floatValue)` method to set the field.

```
transparencyField.set(0.5f);
```

Get the running scene

To get a currently running scene you call:

```
WF3DScene runningScene = WF3DScene.getRunningScene("filename.w3f");
```

Get a named node

To get a named node in the scene graph you call:

```
WF3DNode namedNode = yourWF3DNode.getNode("node name");
```

Get a field

To access a field of a node you call the `getField(int fieldIndex)` method in `WF3DNode`. This method takes as parameter an integer, which indicates what field to retrieve. In the 3D API there is a package named “nodes” containing classes, one for each node type, where constants for all their fields are found.

```
WF3DFi el d yourFi el d = yourWF3DNode. getFi el d(i nt fi el dI ndex);
```

Read and change a field

To read or change a field you use one of the get or set methods found in `WF3DFi el d`. Every `WF3DFi el d` object has a type that shows what X3D/VRML field type it represents, and this type specifies what functions you can call. To learn what X3D/VRML field type a field represents, check the nodes package in the 3D API. In the 3D API description of `WF3DFi el d` there is a conversion table between X3D/VRML types and WireFusion Java types, which is used to see what method calls are applicable. For example, the Boolean java type represents fields of type `SFBool` and you use these two methods to read and change their value:

```
bool ean bool eanVal ue = yourWF3DFi el d. getBool ean();
yourWF3DFi el d. set(bool eanVal ue);
```

Load new textures

To load a texture you call:

```
Obj ect newTexture = WF3DScene. loadI mage(" i mageFi l eName");
```

To replace the texture in a Texture node you would then call:

```
textureNode. getFi el d(Texture. i mage). set(newTexture);
```

Load an external 3D file

To load a new external 3D file to the scene you call:

```
WF3DScene newScene = WF3DScene. loadScene(" fi l eName. w3f");
```

Add and remove a 3D file

To add a new X3D/VRML file you must first load it by calling:

```
WF3DScene newScene = WF3DScene. loadScene(" fi l eName. w3f");
```

To add it to the running scene you do:

```
runni ngScene. addChi l d(newScene);
```

You can remove it by doing:

```
runni ngScene. removeChi l d(newScene);
```

NOTE: You have to convert your X3D/VRML files to the WireFusion 3D Format (file extension `.w3f`) before loading. This is done from the 3DScene object's dialog (Save or Convert buttons).

Replace a 3D file

To replace an X3D/VRML file you do:

```
WF3DScene.replaceScene("filename.w3f", newWF3DScene);
```

Exercises

Exercise: Adding a 3D file to a 3D scene

In this exercise, we will dynamically load and add a 3D teapot to a running 3D scene, which contains a 3D table. This will be achieved using the 3D API and we will place the teapot on the table. We will also learn how to remove the teapot again from the 3D scene.

Step 1 – Load the teapot

Insert a 3DScene object into a new and empty project.

From the **3D Type** dialog, choose to load a **3D Object**.

Load the file *teapot.wrl*, found in the directory *[WireFusion Path]/resources/3D models/teapot/*

NOTE: Remember that any possible changes of the default rendering settings you make for the teapot will not be seen in the final presentation, as it is the master 3D scene's default rendering settings that will be visible, i.e. the table scene, and not the dynamically loaded teapot scene.

Step 2 – Save the teapot in WireFusion 3D format

In the 3DScene dialog, press the **Save** button and store your model on C:\ on your hard disk (Figure 3).

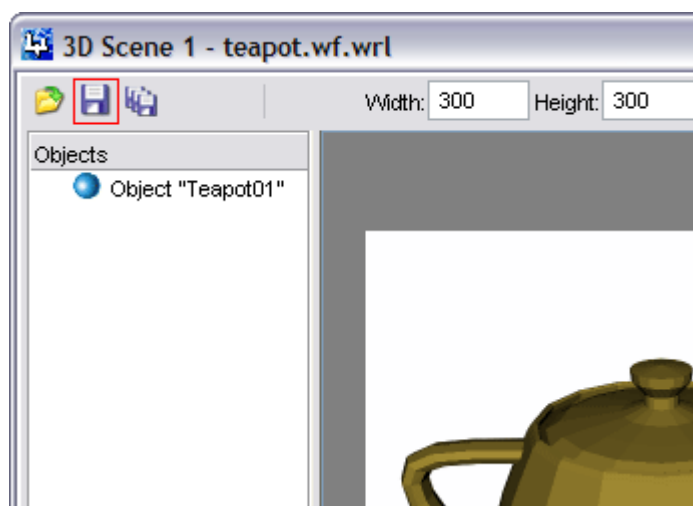


Figure 3: Saving the teapot as .w3f

NOTE: When loading and adding a 3D model dynamically to another 3D scene, using the 3D API, then you first have to manually convert the 3D model (i.e. the model you want to add) to the WireFusion 3D Format (.w3f). This is done by using the save button in the 3DScene dialog.

Step 3 – Load the table

Now, in the 3DScene dialog, press the Replace button (Figure 4) and load the file *table.wrl*, found in the directory *[WireFusion Path]/resources/3D models/table/*

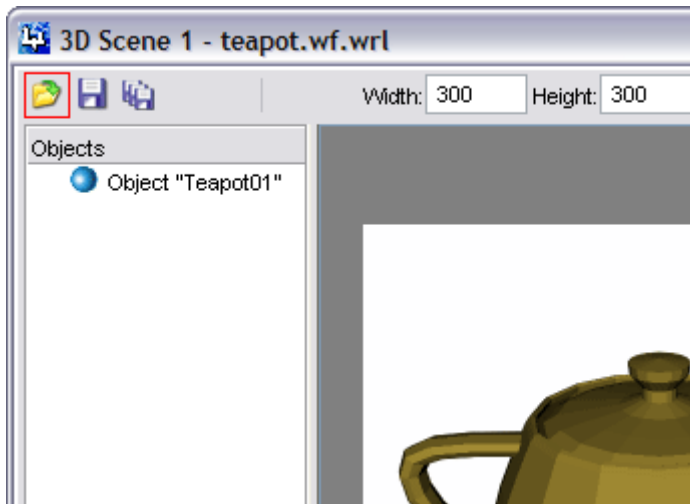


Figure 4: Replacing the teapot model

Make sure to unmark all the checkboxes in the Replace options popup dialog (Figure 5).

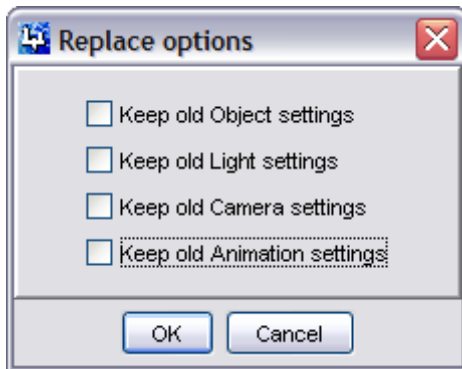


Figure 5: Unmarking all checkboxes in the Replace option dialog

NOTE: You could also have deleted the teapot 3DScene object and inserted a new one, and then loaded the table object. The replacement procedure is not necessary for this project, it is just faster.

Step 4 – Close the table dialog

When the table model is loaded into the 3DScene, without doing any changes, press the OK button to close the 3DScene dialog.

Resize the 3DScene object's Target Area so that it fits the stage dimension.

Press Alt + S on your keyboard to do this automatically.

NOTE: We will use the table scene as the master scene and the default rendering settings you make for the table will also be effective for the 3D scenes you will load/add dynamically.

Step 5 – Insert a Java object and create in-ports

Insert a Java object to your project.

NOTE: To access the 3D node classes, always make sure to import the nodes package when working with the 3D API. See Figure 6.

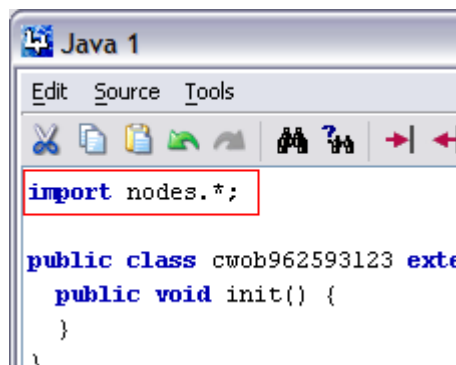


Figure 6: Importing the 3D node classes

In the Java code, insert an in-port. To do this, choose the menu option Tools > Insert In-port code (Figure 7).

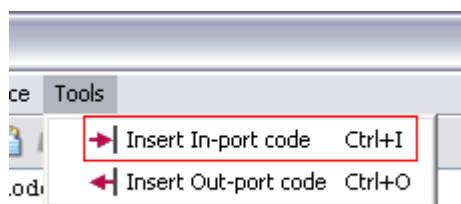


Figure 7: Starting the in-port code generator

When the In-port code generator dialog opens, choose Any Argument as argument, and then choose the port name addTeapot (Figure 8).

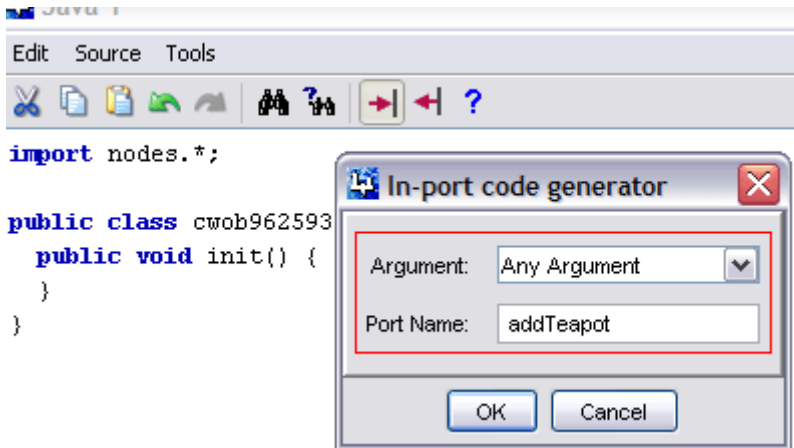


Figure 8: In-port code generator window

Repeat the above and create another in-port, but name this port to removeTeapot instead.

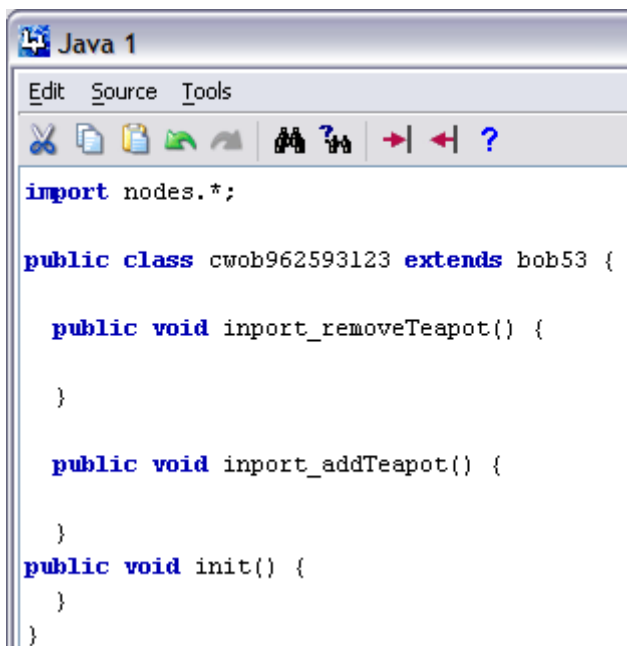


Figure 9: The in-ports added to the code

Step 6 – Adding a scene

Still in the Java object, create a field, named `teapotScene` of type `WF3DScene` as below:

```
private WF3DScene teapotScene;
```

Add the below code into the `inport_addTeapot` method:

```
if (teapotScene != null) return;
WF3DScene tableScene = WF3DScene.getRunningScene("table.w3f");
```

```
teapotScene = WF3DScene.LoadScene("file:///c:/teapot.w3f");
teapotScene.move(0, 85, 0);
teapotScene.scale(0.4f, 0.4f, 0.4f);
tableScene.addChild(teapotScene);
```

- The first line is used to see if we already have added the teapot to the table scene.
- The second line is used to access the running table scene.
- The third line will load the saved teapot scene (Step 2) from your hard drive. We saved it on C:\ to be able to test the functionality from inside WireFusion. You need to replace this line to the correct path before you publish the presentation. If you place the file teapot.w3f in the published project directory, then you should replace the line with:
`teapotScene = WF3DScene.LoadScene("teapot.w3f");`
- The fourth and fifth lines are used to correctly position and scale the teapot scene so it fits into the table scene.
- The sixth and final line will add the teapot scene as a child to the table scene.

Step 7 – Removing a scene

Add the code provided below into the `import_removeTeapot` method:

```
if (teapotScene==null) return;
WF3DScene tableScene = WF3DScene.getRunningScene("table.w3f");
tableScene.removeChild(teapotScene);
teapotScene=null;
```

- The first line is used to see if the teapot is added to the table scene.
- The second line is used to access the running table scene.
- The third line will remove the teapot scene from the table scene.
- The fourth line clears the teapot variable, which indicates that it is not added anymore.

Press the OK button to close the Java object dialog.

Step 8 – Insert Buttons

Insert a Button object. When its dialog opens, choose the button name Add (Figure 10).

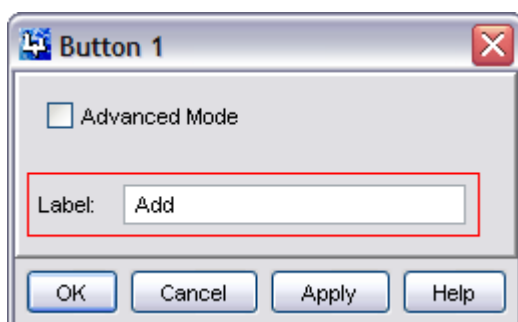


Figure 10: Changing the button label to Add

Insert another Button object. When its dialog opens, choose the button name Remove.

Step 9 – Connect the Button objects to the Java object

Connect:

Button 1 > Out-ports > Button Clicked

to

Java 1 > In-ports > addTeapot

Connect:

Button 2 > Out-ports > Button Clicked

to

Java 1 > In-ports > removeTeapot

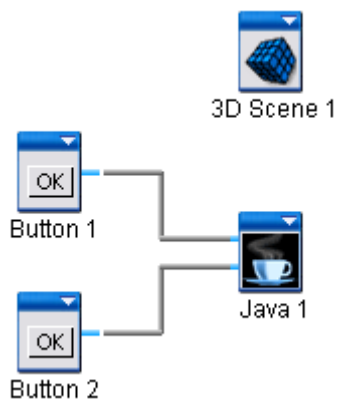


Figure 11: The two Button objects connected to the Java object

Step 10 – Preview

OK, done! If you have followed the steps above, saved the file *teapot.w3f* on *C:* and used the link *file:///c:/teapot.w3f* in your Java code, then it should work to add (and remove) the teapot to (and from) the table.

Press F7 to preview (Figure 12).

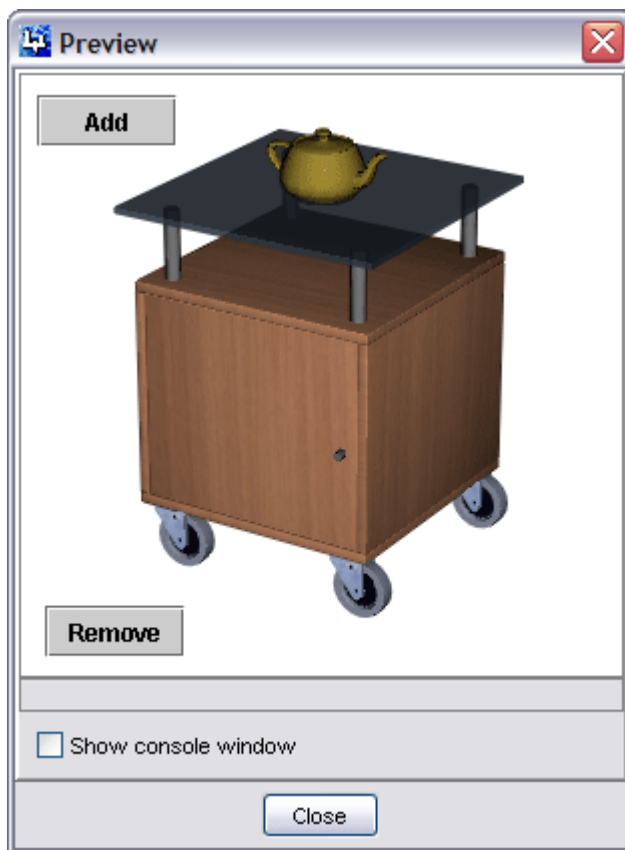


Figure 12: Preview with the teapot added

NOTE: If you publish the presentation, remember to change the link in the code (in step 6, line 3) to `teapot.w3f`, and copy the file to the folder containing your published project.

Exercise: Changing the texture of an object

We will continue with the above example and will now add the possibility to dynamically load and replace a texture, using the 3D API.

Step 1 - Create in-ports

In the Java object, insert a new in-port, Tools > Insert In-port code

When the In-port code generator dialog opens, choose Any Argument as argument, and then choose the port name `changeTexture`.

Step 2 – Replacing the texture

Create a field in your code named `texture` of type `WF3DScene` as below:

```
private Object texture;
```

Add the below code into the `inport_changeTexture` method:

```
WF3DScene tableScene = WF3DScene.getRunningScene("table.w3f");
WF3DNode doorTransformNode = tableScene.getNode("Door");
WF3DNode[] doorChildren = doorTransformNode.getChildren();
WF3DNode shapeNode = null;
for (int i=0; i<doorChildren.length; i++) {
    if (doorChildren[i].type==Node.Shape) {
        shapeNode = doorChildren[i];
        break;
    }
}
WF3DNode appearanceNode = shapeNode.getNode(Shape.appearance);
WF3DNode textureNode = appearanceNode.getNode(Appearance.texture);
texture = WF3DScene.loadImage("file:///c:/ashsen.jpg");
textureNode.getField(Texture.image).set(texture);
```

- The first line is used to access the running table scene.
- The second line retrieves the node named Door.
- The third line retrieves the children nodes of the door Transform node.
- The lines 4-10 scans through the children nodes to retrieve the Shape node under the door transform.
- The 11th line retrieves the Appearance node under the Shape node.
- The 12th line retrieves the Texture node under the Appearance node.
- The 13th line loads the texture from the local hard drive. This line should, before publishing the presentation, be replaced with the line:
`texture = WF3DScene.loadImage("ashsen.jpg");`

- The 14th line retrieves and sets the field image of the Texture node.

Press the OK button to close the Java dialog.

Step 3 – Insert a Button

Insert a Button object. When its dialog opens, choose the button name Texture.

Step 4 – Connect the Button objects to the Java object

Connect:

Button 3 > Out-ports > Button Clicked

to

Java 1 > In-ports > changeTexture

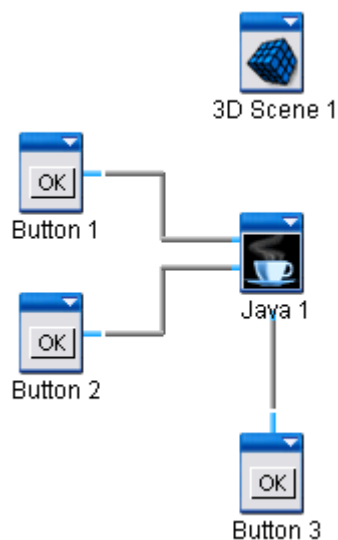


Figure 13: The replace texture button connected to the Java object

Step 5 – Preview

OK, you are done! If you have followed the steps above, have located the file *ashen.jpg* on *C:* and used the link *file:///c:/ashen.jpg* in your Java code, then it should work to replace the texture on the table door.

Press F7 to preview (Figure 14).



Figure 14: Preview with the door texture changed

NOTE: If you publish the presentation, remember to change the link in the code (in step 2, 13th line) to *ashen.jpg*, and copy the file to the folder containing your published project.

Appendix: Supported Nodes and Fields

Out of the 36 X3D/VRML nodes supported in WireFusion, twelve of them are useful when doing advanced 3D programming. Each of these twelve nodes and their supported fields are explained below.

For a full explanation of the X3D and VRML nodes and fields, please refer the X3D and VRML97 specification at the Web3D Consortium web site (www.web3d.org).

Appearance

The Appearance node specifies the visual properties of geometry.

Supported fields in the Appearance node are:

material

The *material* field contains a *Material* node.

texture

The *texture* field contains a *Texture* node.

textureTransform

The *textureTransform* field contains a *textureTransform* node.

DirectionalLight

The DirectionalLight node defines a directional light source that illuminates along rays parallel to a given 3-dimensional vector.

The *direction* field specifies the direction vector of the illumination emanating from the light source in the local coordinate system. Light is emitted along parallel rays from an infinite distance away. A directional light source illuminates only the objects in its enclosing parent group. The light may illuminate everything within this coordinate system, including all children and descendants of its parent group. The accumulated transformations of the parent nodes affect the light.

Supported fields in the DirectionalLight node are:

ambientIntensity

Specifies the ambient intensity, in the range of 0 to 1. The ambient intensity specifies the intensity of the ambient emission from the light.

color

Specifies the light color. Colors are specified with hexadecimal values (0xRRGGBB).

direction

Specifies the light direction.

intensity

Specifies the light intensity, in the range of 0 to 1.

on

Turns the light on and off.

Group

A Group node contains children nodes without introducing a new transformation. It is equivalent to a Transform node containing an identity transform.

Supported fields in the Group node are:

children

Stores the children nodes of this object. Used to access children nodes.

About Grouping Nodes

Grouping nodes have a field that contains a list of children nodes. Each grouping node defines a coordinate space for its children. This coordinate space is relative to the coordinate space of the node of which the group node is a child. Such a node is called a *parent* node. This means that transformations accumulate down the scene graph hierarchy.

Supported grouping nodes:

- Group
- Transform

Supported children nodes:

- DirectionalLight
- Group
- PointLight
- Shape
- TimeSensor
- TouchSensor
- Transform
- Viewpoint

Material

The Material node specifies surface material properties for associated geometry nodes and determines how light reflects off an object to create color.

Supported fields in the Material node are:

ambientIntensity

Specifies the ambient intensity, in the range of 0 to 1. The *ambientIntensity* field specifies how much ambient light from light sources this surface shall reflect. Ambient light is omni directional and depends only on the number of light sources, not their positions with respect to the surface.

diffuseColor

Specifies the material diffuse color. Colors are specified with hexadecimal values (0xRRGGBB). The *diffuseColor* field reflects all light sources depending on the angle of the surface with respect to the light source. The more directly the surface faces the light, the more diffuse light reflects.

emissiveColor

Specifies the material emissive color. Colors are specified with hexadecimal values (0xRRGGBB). The *emissiveColor* field models "glowing" objects.

reflectionMap

The *reflectionMap* field contains a *Texture* node that is used for the reflection. This is field is a WireFusion specific and is not found in X3D/VRML.

shininess

Specifies the material shininess, in the range of 0 to 1. Shininess specifies a material specular scattering exponent.

specularColor

Specifies the material specular color. Colors are specified with hexadecimal values (0xRRGGBB). The *specularColor* and *shininess* fields determine the specular highlights (e.g., the shiny spots on an apple). When the angle from the light to the surface is close to the angle from the surface to the viewer, the *specularColor* is added to the diffuse and ambient colour calculations. Lower shininess values produce soft glows, while higher values result in sharper, smaller highlights.

textureOpacity

Specifies the texture opacity, in the range of 0 to 100. The *opacity* field specifies how "clear" a texture is, with 0 being completely transparent, and 100 completely opaque. This field is a WireFusion specific and is not found in X3D/VRML.

transparency

Specifies the material transparency, in the range of 0 to 1. The *transparency* field specifies how "clear" an object is, with 1 being completely transparent, and 0 completely opaque.

PointLight

The PointLight node specifies a point light source at a 3D location in the local coordinate system. A point light source emits light equally in all directions; that is, it is omnidirectional. PointLight nodes are specified in the local coordinate system and are affected by ancestor transformations.

Supported fields in the PointLight node are:

ambientIntensity

Specifies the ambient intensity, in the range of 0 to 1. The ambient intensity specifies the intensity of the ambient emission from the light.

color

Specifies the light color. Colors are specified with hexadecimal values (0xRRGGBB).

intensity

Specifies the light intensity, in the range of 0 to 1.

location

Specifies the light location.

on

Turns the light on and off.

radius

Specifies the light radius, in the range of 0 to infinity. A PointLight node illuminates geometry within the *radius* of its *location*.

Shape

The Shape node has two fields, *appearance* and *geometry* (not supported), which are used to create rendered objects in the world. The *appearance* field contains an Appearance node that specifies the visual attributes (e.g., material and texture) to be applied to the geometry. The *geometry* field contains a geometry node. The specified geometry node is rendered with the specified appearance nodes applied.

Supported fields in the Shape node are:

appearance

The *appearance* field contains an *Appearance* node.

Texture

The Texture is a merger of the X3D/VRML ImageTexture and PixelTexture nodes.

Supported fields in the Texture node are:

image

Specifies the texture. Takes a WireFusion Texture.

TextureTransform

The TextureTransform node defines a 2D transformation that is applied to texture coordinates. This node affects the way textures coordinates are applied to the geometric surface.

Supported fields in the TextureTransform node are:

center

Specifies the center position for the texture coordinates. The *center* field specifies a translation offset in texture coordinate space about which the *rotation* and *scale* fields are applied.

rotation

Specifies the rotation angle for the texture. The *rotation* field specifies a rotation in radians of the texture coordinates about the *center* point after the scale has been applied. A positive rotation value makes the texture coordinates rotate counterclockwise about the center, thereby rotating the appearance of the texture itself clockwise.

scale

Sets the scale factor for the texture coordinates. The *scale* field specifies a scaling factor of the texture width and height about the *center* point. The *scale* value shall be in the range of negative infinity to positive infinity.

translation

Sets the translation vector of the texture coordinates.

TimeSensor

The TimeSensor node is used to drive animations.

Supported fields in the TimeSensor node are:

fraction

Specifies the animation fraction in the range of 0 to 1. Original X3D/VRML node name is *fraction_changed*.

TouchSensor

A TouchSensor node tracks the location and state of the pointing device and detects when the user points at geometry contained by the TouchSensor node's parent group.

Supported fields in the TouchSensor node are:

enabled

Turns the TouchSensor node on and off. If the TouchSensor node is disabled, it does not track user input or send events.

hitPoint

Contains the 3D point on the surface of the underlying geometry, given in the TouchSensor node's coordinate system. Original X3D/VRML node name is *hitPoint_changed*.

hitTextCoor

Contains the texture coordinates. Original X3D/VRML node name is *hitTextCoord_changed*.

active

Signals TRUE when the primary button is pressed over the TouchSensor and FALSE when it is released. When the field signals TRUE, it grabs all further motion events from the mouse until it is released and sets the *active* field to FALSE (other TouchSensors' will not generate events during this time). Original X3D/VRML node name is *isActive*.

over

Signals True if the mouse cursor is within (or in contact with) the TouchSensor node's geometry. When True it cause the TouchSensor node to generate *active* events (e.g., by pressing the primary mouse button). Original X3D/VRML node name is *isOver*.

Transform

The Transform node is a grouping node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors.

Supported fields in the Transform node are:

center

Specifies a translation offset from the origin of the local coordinate system (0,0,0).

children

Stores the children nodes of this object. Used to access children nodes.

rotation

Specifies a rotation of the coordinate system. The *rotation* field specifies a rotation in radians.

scale

Specifies a non-uniform scale of the coordinate system. *scale* values shall be greater than zero.

scaleOrientation

Specifies a rotation of the coordinate system before the scale (to specify scales in arbitrary orientations). The *scaleOrientation* applies only to the scale operation.

translation

Specifies a translation to the coordinate system.

About Grouping Nodes

Grouping nodes have a field that contains a list of children nodes. Each grouping node defines a coordinate space for its children. This coordinate space is relative to the coordinate space of the node of which the group node is a child. Such a node is called a *parent* node. This means that transformations accumulate down the scene graph hierarchy.

Supported grouping nodes:

- Group
- Transform

Supported children nodes:

- DirectionalLight
- Group
- PointLight
- Shape
- TimeSensor

- TouchSensor
- Transform
- Viewpoint

Viewpoint

The Viewpoint node defines a specific location in the local coordinate system from which the user may view the scene.

Supported fields in the Viewpoint node are:

bound

Specifies the viewpoint when True is received. Original X3D/VRML node name is *set_bind*.

fieldOfView

Specifies the field of view (FOV) value, in the range of 0 to infinity.

orientation

Sets the rotation relative to the default orientation. In the default position and orientation, the viewer is on the Z-axis looking down the -Z-axis toward the origin with +X to the right and +Y straight up.

position

Sets the position relative to the coordinate system's origin (0,0,0). In the default position and orientation, the viewer is on the Z-axis looking down the -Z-axis toward the origin with +X to the right and +Y straight up.

Index

3

3D API	
Basics	6
Description	1

A

Appearance Fields	
material	21
texture	21
textureTransform	21
Appearance Node	21

D

Definitions	
W3F	3, 4
VRML	3
X3D	3
DirectionalLight Fields	
ambientIntensity	21
color	22
direction	22
intensity	22
DirectionalLight Node	21

F

Fields	3, 5
Appearance	21
DirectionalLight	21
Group	22, 23
PointLight	24
Shape	25
Texture	25
TextureTransform	25
TimeSensor	26
TouchSensor	26
Transform	27
Viewpoint	28

G

Group Fields	
children	22
Group Node	22
Grouping Nodes	22

M

Material Fields	
ambientIntensity	23
diffuseColor	23
emissiveColor	23
reflectionMap	23
shininess	23
specularColor	23
textureOpacity	24
transparency	24
Material Node	23

N

Node Types	3, 4
Nodes	3, 4
Appearance	21
DirectionalLight	21
Group	22
Material	23
PointLight	24
Shape	25
Texture	25
TextureTransform	25
TimeSensor	26
TouchSensor	26
Transform	27
Viewpoint	28

P

PointLight Fields	
ambientIntensity	24
color	24
intensity	24
location	24
on	24
radius	24
PointLight Node	24

R

Requirements	2
System Requirements	2
User Requirements	2

S

Scene Graph	3, 4
Shape Fields	
appearance	25
Shape Node	25
System Requirements	2

T

Texture Fields	
image	25
Texture Node	25
TextureTransform Fields	
center	25
rotation	25
scale	25
translation	26
TextureTransform Node	25
TimeSensor Fields	
fraction	26
TimeSensor Node	26
TouchSensor Fields	
active	26
enabled	26
hitPoint	26

hitTextCoord	26	Viewpoint Fields	
over	26	bound	28
TouchSensor Node	26	fieldOfView	28
Transform Fields		orientation	28
center	27	position	28
children	27	Viewpoint Node	28
rotation	27	WireFusion 3D Format	3
scale	27	VRML	
scaleOrientation	27	Fields	3
translation	27	Node Types	3
Transform Node	27	Nodes	3
U		Scene Graph	3
User Requirements	2	X	
V,W		X3D	
W3F		Fields	3
Fields	5	Node Types	3
Node Types	4	Nodes	3
Nodes	4	Scene Graph	3
Scene Graph	4		

* * *

Publication date: December 14, 2005

The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

All rights reserved © 2005 Demicron AB, Sundbyberg, Sweden. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Demicron and its licensors, if any.

Trademarks — Demicron and WireFusion are trademarks of Demicron AB. Acrobat, Photoshop are registered trademarks of Adobe Systems Incorporated. Java, SunSoft are trademarks of Sun Microsystems, Inc. Windows95, Windows98, Windows ME, Windows NT, Windows 2000, Windows XP are trademarks or registered trademarks of Microsoft Corporation. Pentium is a trademark of Intel Corporation. OS X is a trademark of Apple Computer. All other trademarks are the property of their respective owners.