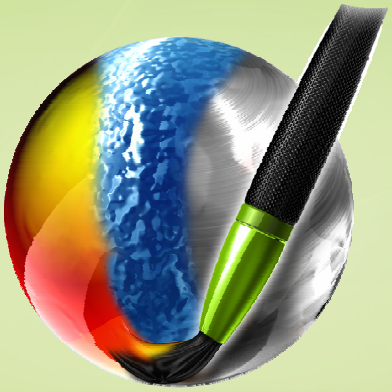
Semantic and Annotation Remapping in FX Composer



April 2008

Table of Contents

Semantic and Annotation Remapping

FX Composer includes a semantic remapper to allow developers to fine-tune the inputs that FX Composer will feed their effects.

In previous versions only a name remapping was possible, but FX Composer now supports complete type remapping. For example, if your effects have special need of a forged matrix or vector, FX Composer now allows you to manipulate the default fixed semantics to make the input you want.

Numerous operators like Matrix/Vector operations are shipped with the product. This collection of operators can be extended by users using the FX Composer plugin system, meaning you can code the operator that suits your needs.

Syntax

FX Composer remappings are located in a file named “mappings.xml” located in <application root>\Plugins\scene\render\Data\.

If you open this file, you will notice that it contains several predefined remappings. We’ve included as many mappings so you don’t have to create them yourself.

The system is based on a graph of what we call “operator nodes.” These nodes are connected to form a graph that will have inputs from FX Composer and that will have an output calculated from the traversed nodes.

Here is a simple example of a Matrix Multiplication:

```
1: <Remapping name="Name">
2:      <identifiers>
3:           <parametername value="WorldView"/>
4:      </identifiers>
5:      <expression>
6:           <MatrixMultiply description="World * View">
7:                <input type="internalsemantic" value="world"/>
8:                <input type="internalsemantic" value="view"/>
9:           </MatrixMultiply>
10:     </expression>
11: </Remapping>
```

The Remapping Tag (line 1):

This tag specifies a block of remapping. The name parameter can be set to anything helpful to document what the remapping does.

The identifiers: (Lines 2 to 4)

This tag defines what needs to be matched in the shader file in order for this remapping to be used. In the previous Matrix Multiplication example, the only thing that we need in order to remap this parameter is that its name be ViewInv.

Other identifiers can be used such as:

<semantic name="LightPosition"/>

This option specifies that in order to be remapped, the name of the parameter semantic has to match.

<parametername value="LightPos"/>

This option specifies that the parameter has to match the name of the parameter itself.

<annotation name="object" value="PointLight0"/>

This option specifies that the parameter has to have the specific annotation/value pair in order to match.

All of these identifiers can be used at the same time, but the remapping will be active only if all the identifiers match.

An Expression Node (Line 5):

This node specifies the beginning of an expression block containing remapping operators.

An Operator node (line 6):

In this case, “MatrixMultiply” represents a node that will have two inputs and one output. The inputs are the two child nodes of MatrixMultiply.

Other types of operators include MatrixTranspose, MatrixInverse, DotProduct, CrossProduct, Cosine, Sine, Log, Ceiling, and MatrixSelect…

All operator nodes have a Description attribute that can be used for debugging. This description attribute is used when an error or warning is displayed by FX Composer. This allows you to debug and locate any remapping problems.

Two Input Nodes (line 7 and 8):

This operator node is created at runtime and two of its inputs come from FX Composer Internals. These internals are referenced using the input tag with a type set to “internalsemantic,” telling FX Composer that this node will in fact be fed with an internal value. In our example, the world and view matrix.

The output of the operator is then used by its parent if it is a node, or is directly fed to the shader if it is the last node.

Operator nodes can be linked to make even more complex remappings. A list of complete examples is featured at the end of this user reference.

List of Operators

MatrixMultiply

Description

Used to multiply two matrices.

Input/Output

Direction	Type	Remarks
Input 1	Matrix	Left matrix
Input 2	Matrix	Right matrix
Output	Matrix	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<MatrixMultiply description=”World * View”>
        <input type="internalsemantic" name="world"/>
        <input type="internalsemantic" name="view"/>
</ MatrixMultiply >

<MatrixMultiply description="(World * View) * Projection”/>
        <MatrixMultiply description="World * View">
                <input type="internalsemantic" name="world"/>
                <input type="internalsemantic" name="view"/>
        </MatrixMultiply>
        <input type="internalsemantic" name="projection"/>
</MatrixMultiply>
```

MatrixAdd

Description

Used to add two matrices.

Input/Output

Direction	Type	Remarks
Input 1	Matrix	Left Matrix
Input 2	Matrix	Right Right
Output	Matrix	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<MatrixAdd description=”World + View”>
        <input type="internalsemantic" name="world"/>
        <input type="internalsemantic" name="view"/>
</ MatrixAdd >
```

MatrixSubstract

Description

Used to substract two matrices.

Input/Output

Direction	Type	Remarks
Input 1	Matrix	Left Matrix
Input 2	Matrix	Right Right
Output	Matrix	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<MatrixSubstract description=”World - View”>
        <input type="internalsemantic" name="world"/>
        <input type="internalsemantic" name="view"/>
</ MatrixSubstract >
```

MatrixInverse

Description

Used to get the inverse of a matrix.

Input/Output

Direction	Type	Remarks
Input	Matrix	
Output	Matrix	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<MatrixInverse>
        <input type="internalsemantic" name="view"/>
</MatrixInverse>
```

MatrixTranspose

Description

Used to get the transpose of a matrix.

Input/Output

Direction	Type	Remarks
Input	Matrix	
Output	Matrix	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<MatrixTranspose>
        <input type="internalsemantic" name="view"/>
</MatrixTranspose>
```

MatrixTransposeConditional

Description

Used to get the transpose of a matrix only of Input 2 is true.

Input/Output

Direction	Type	Remarks
Input 1	Matrix	
Input 2	Boolean	The transpose will only be returned if this input is true
Output	Matrix	

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
<MatrixTransposeConditional>
        <input type="internalsemantic" name="view"/>
        <input type=”internalsemantic” name=”isopengl”/>
</ MatrixTransposeConditional >
```

MatrixScale

Description

Used to multiply a matrix by a scalar factor.

Input/Output

Direction	Type	Remarks
Input 1	Matrix	
Input 2	float	
Output	Matrix	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<MatrixScale description="Matrix Scale">
        <input type="internalsemantic" value="world"/>
        <input type="float" value="0.1"/>
</MatrixScale>
```

MatrixOrthonormalize

Description

Used to orthonormalize a matrix.

Input/Output

Direction	Type	Remarks
Input	Matrix	
Output	Matrix	

PParameters

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
<MatrixOrthonormalize>
        <input type="internalsemantic" name="view"/>
</MatrixOrthonormalize >
```

MatrixSelect

Description

Used to select a row/column of a matrix.

Input/Output

Direction	Type	Remarks
Input	Matrix	
Output	Vector	

Name	Value	Remarks
Description		Used for error handling
Mode	Row1-4, Column1-4	Options are not case sensitive

Parameters

Remarks

The output vector will the size of the column/row selected.

Example

```
<MatrixSelect mode="Row4">
        <input type="internalsemantic" name="view"/>
</MatrixSelect>
```

VectorMultiply

Description

Used to multiply a vector by a scalar.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Input 2	float	
Output	Vector	

Parameters:

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
<RemappedSemantic name="myVectorMultiplication">
        <VectorMultiply>
                <input type="float3" value="1, 0, 0"/>
                <input type="float" value="2.0"/>
        </VectorMultiply>
</RemappedSemantic>
```

VectorAdd

Description

Used to add two vectors.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Input 2	Vector	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<RemappedSemantic name="myfloat3Addition">
        <VectorAdd>
                <input type="float3" value="1, 0, 0"/>
                <input type="float3" value="1, 0, 0"/>
        </VectorAdd>
</RemappedSemantic>
```

VectorSubtract

Description

Used to subtract two vectors.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Input 2	Vector	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<VectorSubtract>
        <input type="float3" value="1, 0, 0"/>
        <input type="float3" value="1, 0, 0"/>
</ VectorSubtract >
```

VectorLength/VectorLengthSq

Description

Used to get the length of a vector.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Output	Float	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<VectorLength>
        <input type="float3" value="1, 0, 0"/>
</ VectorLength >


<VectorLengthSq>
        <input type="float4" value="1, 0, 0, 0.25"/>
</ VectorLengthSq >
```

VectorNornalize

Description

Used to normalize a vector.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<VectorNormalize>
        <input type="float3" value="1, 0, 0"/>
</VectorNormalize >
```

CrossProduct

Description

Used to calculate the cross product of two vectors.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Input 2	Vector	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<CrossProduct>
        <input type="float3" value="1, 0, 0"/>
        <input type="float3" value="0, 1, 0"/>
</CrossProduct >
```

DotProduct

Description

Used to calculate the dot product of two vectors.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Input 2	Vector	
Output	Float	

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
<DotProduct>
        <input type="float3" value="1, 0, 0"/>
        <input type="float3" value="0, 1, 0"/>
</DotProduct>
```

Cast

Description

Used to get the cast from a type to another.

Input/Output

Direction	Type	Remarks
Input	Vector	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling
To	Float2, float3, float4	

Example

```
<Cast to="float3">
        <MatrixSelect mode="Row4">
                <MatrixInverse>
                        <input type="internalsemantic" name="view"/>
                </MatrixInverse>
        </MatrixSelect>
</Cast>
```

TransformCoordinate

Description

Used to transform a coordinate value by a matrix.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Input 2	Matrix	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<TransformCoordinate>
        <input type="float3" value="0, 1, 0"/>
        <input type="internalsemantic" value="world"/>
</TransformCoordinate>
```

TransformNormal

Description

Used to transform a normal value by a matrix.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Input 2	Matrix	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<TransformNormal>
        <input type="float3" value="0, 1, 0"/>
        <input type="internalsemantic" value="world"/>
</ TransformNormal >
```

VectorSetValue

Description

Used to set the value of an individual component of a vector.

Input/Output

Direction	Type	Remarks
Input 1	Vector	The input vector to modify
Input 2	Float	The value to assign to the selected component
Output	Vector	The modified vector

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling
Operation	Char	Examples: "x" "y" "a" …

Remarks

This operator creates a vector of the length of operation parameter. It then fills the vector, component by component based on the order the parameters.

Example

```
<VectorSetValue component="z">
        <input type="float3" value="1, 2, 3"/>
        <input type="internalsemantic" value="currenttime"/>
</VectorSetValue>
```

This will result in an output of: "1, 2, 1233.021321" (last number being the current time).

Swizzle

Description

Used to transform a normal value by a matrix.

Input/Output

Direction	Type	Remarks
Input 1	Vector	
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling
Operation	String	Examples: xyz yxwz argb rgba …

Remarks

This operator creates a vector of the length of operation parameter. It then fills the vector; component by component based on the order of the parameters.

Examples

```
<Swizzle operation="yxz">
        <input type="float3" value="1, 2, 3"/>
</Swizzle>
```

This will result in an output of: “2, 1, 3.”

```
<Swizzle operation="wwyz">
        <input type="float4" value="1, 2, 3, 4"/>
</Swizzle>
```

This will result in an output of: “4, 4, 2, 3.”

Demux

Description

Used to separate a vector into individual outputs.

Input/Output

Direction	Type	Remarks
Input	Vector	
Output	*n* individual pins	*n* being the size of the vector

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling
Ooutputsize	Number (1, 2, 3, 4)	Determines the number of output pins to create from the input vector. It usually is the dimension of the vector.

Example

```
<DeMux outputsize="2">
        <MatrixSelect mode="Column1">
                <input type="internalsemantic" value="world"/>
        </MatrixSelect>
</DeMux>
```

Mux

Description

Used to concatenate output pins into a single vector of a given dimension.

Input/Output

Direction	Type	Remarks
Input	*n* individual pins	*n* being the size of the vector
Output	Vector	

Parameters

Name	Value	Remarks
Description		Used for error handling
Inputsize	Number (1, 2, 3, 4)	Determines the number of inputs that the parser will have to look for. This will also be the size of the output vector.

Example

Creating a float3 with currenttime, lasttime and a constant:

```
<Mux inputsize="3">
        <input type="internalsemantic" value="currenttime"/>
        <input type="internalsemantic" value="lasttime"/>
        <input type="float" value="0.025"/>
</Mux>
```

Cosine/ACosine/Sine/ASine/Tangent/ATangent

Description

This operator is used to calculate the sine, cosine and tangent of the input value.

Input/Output

Direction	Type	Remarks
Input	Float, double	
Output	Float	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<Cosine>
        <input type="internalsemantic" value="currenttime"/>
</Cosine>

<Tangent>
        <input type="internalsemantic" value="currenttime "/>
</Tangent>
```

Log

Description

Returns the logarithm of a number.

Input/Output

Direction	Type	Remarks
Input	Float	
Output	Float	

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
<Log>
        <input type="internalsemantic" value="currenttime"/>
</Log>
```

Pow

Description

Returns x raised to the power of y.

Input/Output

Direction	Type	Remarks
Input1	Float	
Input2	Float	Power
Output	Float	

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
<Pow>
        <input type="internalsemantic" value="currenttime"/>
        <input type="float" value="2.0"/>
</Pow>
```

Add/Substract/Multiply/Divide/Modulus

Description

Returns performed operation.

Input/Output

Direction	Type	Remarks
Input1	Float	
Input2	Float	
Output	Float	

Parameters

Name	Value	Remarks
Description		Used for error handling

Example

```
<Add>
        <input type="internalsemantic" value="currenttime"/>
        <input type="float" value="2.0"/>
</Add>

<Substract>
        <input type="internalsemantic" value="currenttime"/>
        <input type="float" value="2.0"/>
</Substract>
```

Floor

Description

Returns the largest integer less than or equal to the specified decimal number.

Input/Output

Direction	Type	Remarks
Input	Float	
Output	Float	

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
< Floor >
        <input type="internalsemantic" value="currenttime"/>
</ Floor >
```

Ceiling

Description

Returns the smallest integer greater than or equal to the specified decimal.

Input/Output

Direction	Type	Remarks
Input	Float	
Output	Float	

Parameters

Name	**Value**	**Remarks**
Description		Used for error handling

Example

```
< Ceiling >
        <input type="internalsemantic" value="currenttime"/>
</ Ceiling >
```

Input (input)

Description

This node is used to reference an internal FX Composer 2.5 Semantic.

Input/Output

Direction	Type	Remarks

Parameters

Name	Value	Remarks
Type	Internalsemantic	
Value	World, view, projection, isopengl, mouseposition, time, lasttime, elapsedtime, random	

Name	Value	Remarks
Type	Float	
Value	Floating point value, pi	

Name	Value	Remarks
Type	Float2, float3, float4	
Value	Floating point values separated by comas	

Name	Value	Remarks
Type	Light, camera, node	The type of object you want to bind this semantic to
Name	String	Any group name that will represent the light, camera or object node you want to link
Value	Position, direction, color	The parameter you want to link
Space	World, object, view	Default is world

Examples

```
<MatrixTranspose>
        <input type="internalsemantic" name="view"/>
</MatrixTranspose>


<input type="float3" value="1, 2, 3"/>
```

Programming Your Own Operator Nodes

FX Composer 2.5 gives you the opportunity to add operator nodes that suit your needs. The concept used is the same as for regular FX Composer plugins.

The remapping is based on a graph composed of Remapping Operator Nodes.

- Each node has inputs and outputs that can be of various types. (FXMatrix, double, float…)
- Each input and output of these operators are connected to other operators.
- Every operator node has to inherit from FXSemanticMappingNode and implement the IFXPlugin Interface.

Integration into FX Composer

The operator nodes are loaded into FX Composer 2.5 using an fxplug file. Refer to main documentation for more information.

Here is an example of a declaration:

```
<virtualdirectory path="FX Composer/remappingoperators">
        <class name=" DummyNamespace.VectorLength"/>
</virtualdirectory>
```

Note that the virtual directory of the operator nodes has to be : “FX Composer/remappingoperators”.

FX Composer will look for operators inside this virtual directory.

Naming Convention

FX Composer semantics remapping relies on string comparisons to figure out which operator to create from the xml nodes inside the file. Therefore a naming convention has been established. An operator node must have a name ending by “Node”.

The name inside the xml is used without the “Node” suffix.

For the case below, the xml file would look like:

```
<VectorLength>
…
</VectorLength>
```

FX Composer then adds the suffix and looks for the type of the operator in its database.

Simple Example: VectorLength

In this case, this operator only takes one input and outputs one value.

```
public class VectorLengthNode : FXSemanticMappingNode
{
```

```
        private FXProperty<float> _Output;
        public override IFXProperty Output
        {
                get { return _Output; }
        }

        private FXProperty<FXMatrix> _Input;
        public override IFXProperty Input
        {
                get { return _Input; }
        }

        public VectorLengthNode()
        {
                _Output = new FXProperty<float>(this.Properties, "Length", "");
                _Input = new FXProperty<FXMatrix>(this.Properties, "Input Vector", "");
                _Input.PropertyValue = new FXMatrix(1, 3);
                _Output.DependsOn(_Input);
        }

        public override void EvaluateProperties()
        {
                FXMatrix mat = _Input.PropertyValue;
                FXVector3 vector = new FXVector3((float)mat[0, 0], (float)mat[0, 1],
(float)mat[0, 2]);
                _Output.PropertyValue = vector.Length();
        }
}
```

Remarks:

The constructor creates the properties and assigns a default value to the input.

The Evaluate properties method is called whenever this operator node has to be used.

Input and Output properties are used to access the internal fields.

Case of Multiple Inputs/Output

In the following case, a Matrix Scale, the operator needs two inputs. For that case, a property called InputPins is overridden from the base class. The getter of this property returns an array of IFXProperty containing the inputs in the order that they should appear in the xml file.

In the case below, it creates an array containing the internal input matrix, and the scalar value. They should appear in the same order in the mapping.xml file.

Example:

```
<MatrixScale description="Matrix Scale">
        <input type="internalsemantic" value="world"/>
        <input type="float" value="0.1"/>
</MatrixScale>
```

Here is the code for a MatrixScaleNode:

```
public class MatrixScaleNode : FXSemanticMappingNode, IFXPlugin
{
        private FXProperty<FXMatrix> _Output;
        public override IFXProperty Output
        {
                get { return _Output; }
        }

        private FXProperty<double> _Scalar;
        public FXProperty<double> Scalar
        {
                get { return _Scalar; }
                set { _Scalar = value; }
        }

        private FXProperty<FXMatrix> _Input;

        public override IFXProperty[] InputPins
        {
                get
                {
                        return new IFXProperty[] { _Input, _Scalar};
                }
        }

        public MatrixScaleNode()
        {
                _Output = new FXProperty<FXMatrix>(this.Properties, "Output Vector",
"");
                _Input = new FXProperty<FXMatrix>(this.Properties, "Input Matrix", "");
                _Input.PropertyValue = FXMatrix.Identity4x4;
                _Scalar = new FXProperty<double>(this.Properties, "Scalar", "");
                _Output.DependsOn(_Input);
        }

        public override void EvaluateProperties()
        {
                FXDebug.CheckArgument(_Input.PropertyValue, "Input pin cannot be
null");
                double scalar = _Scalar.PropertyValue;
                FXMatrix mat = _Input.PropertyValue * FXMatrix.Scaling((float)scalar,
                                                                        (float)scalar,
                                                                        (float)scalar);
                _Output.PropertyValue = mat;
        }

        #region IFXPlugin Members
        public bool Build(IFXPluginType pluginType)
```

```
        {
            return true;
        }
        #endregion

}
```

Complete Examples

Retrieving the Camera Position from the View Matrix

```
<Remapping name="">
    <identifiers>
        <semantic value="myCameraPosition"/>
    </identifiers>
    <expression>
        <Cast to="float3">
            <MatrixSelect mode="Row4">
                <MatrixInverse>
                    <input type="internalsemantic" value="view"/>
                </MatrixInverse>
            </MatrixSelect>
        </Cast>
    </expression>
</Remapping>
```

Calculating the World x View x Projection Matrix

```
<Remapping name="">
    <identifiers>
        <semantic value="myWorldViewProjection"/>
    </identifiers>
    <expression>
        <MatrixTransposeConditional description="">
            <MatrixMultiply description="(World * View) * Projection">
                <MatrixMultiply description="World * View">
                    <input type="internalsemantic" value="world"/>
                    <input type="internalsemantic" value="view"/>
                </MatrixMultiply>
                <input type="internalsemantic" value="projection"/>
            </MatrixMultiply>
            <input type="internalsemantic" value="isopengl"/>
        </MatrixTransposeConditional>
    </expression>
</Remapping>
```

Calculating the World Inverse Transpose

```
<Remapping name="">
	<identifiers>
		<semantic value="myWorldInverseTranspose"/>
	</identifiers>
	<expression>
		<MatrixTransposeConditional description="OGL Render devices">
			<MatrixTranspose description=" ">
				<MatrixInverse description=" ">
					<input type="internalsemantic" value="world"/>
				</MatrixInverse>
			</MatrixTranspose>
			<input type="internalsemantic" value="isopengl"/>
		</MatrixTransposeConditional>
	</expression>
</Remapping>
```

Computing the Angle Between Two Vectors in Degrees

```
<Remapping name="">
	<identifiers>
		<semantic value="myVectorAngleDegrees"/>
	</identifiers>
	<expression>
		<Divide>
			<Multiply>
				<ACosine>
					<DotProduct>
						<VectorNormalize>
							<input type="float3" value="1, 1, 0"/>
						</VectorNormalize>
						<input type="float3" value="0, 1, 0"/>
					</DotProduct>
				</ACosine>
				<input type="float" value="180"/>
			</Multiply>
			<input type="float" value="pi"/>
		</Divide>
	</expression>
</Remapping>
```

Computing the Angle Between Two Vectors in Radians

```
<Remapping name="">
        <identifiers>
                <semantic value="myVectorAngleRadians"/>
        </identifiers>
        <expression>
                <ACosine>
                        <DotProduct>
                                <VectorNormalize>
                                        <input type="float3" value="1, 1, 0"/>
                                </VectorNormalize>
                                <input type="float3" value="0, 1, 0"/>
                        </DotProduct>
                </ACosine>
        </expression>
</Remapping>
```

Scripting

FX Composer supports scripting through any compliant .NET language using plugins. FX Composer contains a Python scripting window that can be used to control the engine using Python. A sample command list might be:

```
from fxcapi import *
Reset()
Teapot()
Translate(1,1,1)
Teapot();
Undo()
Undo();
Undo()
Redo();
import FXComposer.Scene
dir(FXComposer.Scene)
```

Scripting can be used for task automation, interprocess communication, changes and additions to the UI, and more. For information on the “IronPython” version of Python used in FX Composer 2, see the IronPython community Web page at http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython

List of Commands

The complete list of all the FX Composer commands can be found in FX Composer 2.5 API Documentation document (docs/SDK/FXComposer2 API.htm). It can be easily accessed from the Start Menu as well.

Scripting Toolbars

FX Composer allows you to create custom toolbars with buttons that execute Python scripts. For more information, please see the section on Working with Layouts.

Sample Scripts

Various use cases:

- Convert CgFX/.fx to COLLADA FX
- Binding light objects to parameters
- Assigning shaders based on name

A set utility scripts can be found in the FX Composer 2.5 SDK section called **Error! Reference source not found.**, page **Error! Bookmark not defined.**.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com