



USER GUIDE

VERSION 2.1V3

Katana™ User Guide. Copyright © 2015 The Foundry Visionmongers Ltd. All Rights Reserved. Use of this document and the Katana software is subject to an End User License Agreement (the "EULA"), the terms of which are incorporated herein by reference. This document and the Katana software may be used or copied only in accordance with the terms of the EULA. This document, the Katana software and all intellectual property rights relating thereto are and shall remain the sole property of The Foundry Visionmongers Ltd. ("The Foundry") and/or The Foundry's licensors.

The EULA can be read in the Katana User Guide.

The Foundry assumes no responsibility or liability for any errors or inaccuracies that may appear in this document and this document is subject to change without notice. The content of this document is furnished for informational use only.

Except as permitted by the EULA, no part of this document may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, recording or otherwise, without the prior written permission of The Foundry. To the extent that the EULA authorizes the making of copies of this User Guide, such copies shall be reproduced with all copyright, trademark and other proprietary rights notices included herein. The EULA expressly prohibits any action that could adversely affect the property rights of The Foundry and/or The Foundry's licensors, including, but not limited to, the removal of the following (or any other copyright, trademark or other proprietary rights notice included herein):

Katana™ software © 2015 The Foundry Visionmongers Ltd. All Rights Reserved. Katana™ is a trademark of The Foundry Visionmongers Ltd.

Sony Pictures Imageworks is a trademark of Sony Pictures Imageworks.

Mudbox™ is a trademark of Autodesk, Inc.

RenderMan ® is a registered trademark of Pixar.

In addition to those names set forth on this page, the names of other actual companies and products mentioned in this Reference Guide (including, but not limited to, those set forth below) may be the trademarks or service marks, or registered trademarks or service marks, of their respective owners in the United States and/or other countries. No association with any company or product is intended or inferred by the mention of its name in this document.

Linux ® is a registered trademark of Linus Torvalds.

The Foundry

5 Golden Square,

London,

W1F 7HT

Rev: 16 June 2016

CONTENTS

Preface

Key Concepts	19
User Interface and Naming Conventions	20
Getting Help	20
Viewing Additional Help	20
Contacting Customer Support	21
Sending Crash Reports	21

Installation and Licensing

System Requirements	23
Supported Renderers	24
Install Katana	24
KATANA_RESOURCES	25
Connecting Katana to a Renderer	25
Changing the Default Renderer	26
Arnold Specific Notes	26
PRMan Specific Notes	26
Network Configuration	27
Python Search Path	27
Launching Katana	27
Licensing Katana	28
About Licenses	28
Setting up the Floating License Server	28
Setting up the License on the Client Machine	29
Further Information	29
Configuring Message Level	29
Defining the directory in which Katana stores temporary files	30

The Way of The Katana

Katana Introduction	31
Making a Scene	32

Customizing Your Workspace

Workspace Overview	46
The Default Workspace	46
The Default Tabs	47
Menu Bar Components	47
Customizing Your Workspace	50
Adjusting Layouts	51
Saving Layouts	53
Loading Layouts	53
Deleting Layouts	53
Custom Render Resolutions	53
Using the UI	53
Preferences	54
Application	54
Color	57
Dopesheet	57
ExternalTools	58
Layout	58
Monitor	59
NodeTags	60
Nodegraph	60
Nodes	62
Parameters	62
Python	63
Scenegraph	64
Viewer	64
Creating a Katana Project	
Katana Projects and Recipes	65
Creating a New Katana Project	65
Saving a Katana Project	65
Loading a Katana Project	66
Importing a Katana Project	67
Exporting a Katana Project	67
Changing a Project's Settings	68
Dealing With Assets	69
Selecting an Asset Manager	70
Using the File Browser	70
Autosaves	71
Working with Nodes	
Adding Nodes	73
Selecting Nodes	74
Connecting Nodes	76

Connecting a Node into the Recipe	76
Removing a Node from the Recipe	77
Tidying the Recipe with a Dot Node	77
Replacing Nodes	78
Copying and Pasting Nodes	78
Cloning Nodes	79
Disabling Nodes	79
Deleting Nodes	80
Navigating Inside the Node Graph	80
Panning	80
Zooming	80
Fitting Selected Nodes in the Node Graph	81
Fitting the Node Tree in the Node Graph	81
Organizing Nodes in Group Nodes	81
Creating Group Nodes	81
Navigating in Hierarchies of Group Nodes	82
Connecting Group Nodes	83
Editing Group Nodes	83
Node Buttons	84
Node Button Keyboard Shortcuts	84
Improving Readability and Navigation with Backdrop Nodes	85
Creating a Backdrop Node	85
Editing a Backdrop Node	85
Editing a Node's Parameters	87
Accessing a Node's Parameters	88
Opening and Closing a Node's Parameters	88
Default Parameters Tab Icons	89
Changing a Node's Name	89
Changing a Node's Parameters	89
State Badges	91
Customizing Node Display	92
Changing a Node's Background Color	92
Dimming Nodes not Connected to the View Node	92
Displaying Nodes Linked by an Expression	93
Drawing the Node Graph with Reduced Contrast	93
Aiding Project Readability with Node Icons	93
Image Thumbnails	94
Indicators on Nodes	94
Using the Scene Graph	
Scene Graph Terminology	96

Viewing the Scene Graph	97
Navigating the Scene Graph History	97
Manipulating the Scene Graph	97
Selecting and Deselecting Locations in the Scene Graph	98
Selecting Locations with the Search Facility	99
Expanding the Scene Graph	99
Collapsing the Scene Graph	100
Bookmarking a Scene Graph State and Working Sets	101
Viewing a Location's Attributes	102
Changing What is Shown in the Viewer	103
Rendering only Selected Objects	104
Controlling Live Rendering in the Scene Graph	104
Using the Live Render Updates Column	105
Using the 3D Update Mode	105
Turning on Implicit Resolvers	105
Making Use of Different Location Types and Proxies	106
Using Assemblies and Components	106

Working Sets

Working Sets in Scene Graph Tab	107
Interacting with Working Sets	109
Saving and Restoring Working Sets States	110

Adding 3D Assets

Overview	111
Adopting Alembic	112
Adding an Alembic Asset	112
Describing an Asset Using XML	112
Adding an Asset Described Using XML	113
Using the Importomatic	113
Adding Assets Using the Importomatic	113
Editing an Importomatic Asset's Parameters	114
Editing an Asset's Name	114
Disabling an Asset	115
Enabling an Asset	115
Deleting an Asset from the Importomatic	115
Assigning a Look File to an Asset	116
Assigning an Attributes File to an Asset	116
Adding Additional Outputs to the Importomatic	116

Changing an Output's Name	116
Generating Scene Graph Data with a Plug-in	117
Adding a Scene Graph Generator Location to Your Scene Graph	117
Forcing the Scene Graph Generator to Execute	118

Adding and Assigning Materials

Overview	120
Creating a Material	121
Adding a Shader to a Material Location	121
Creating a Material from a Look File	122
Creating a Material that is a Child of Another Material	122
Editing a Material	123
Overriding a Material	123
Multiple Materials with a MaterialStack Node	124
Adding a Material	124
Adding a Material from a Look File	124
Adding a Material as a Child	124
Duplicating a Material	125
Disabling a Material	125
Deleting a Material	125
Adding a Material Node from the Node Graph	125
Moving Materials Within the Add List	125
Incorporating PRMan Co-Shaders	126
Network Materials	127
Creating a Network Material	127
Using a Network Shading Node	130
Creating a Network Material's Public Interface	133
Changing a Network Material's Connections	135
Editing a Network Material	136
Assigning Materials	137
Assigning Textures	138
Using Face Sets	138
Creating a Face Set	138
Editing a Face Set	139
Assigning Materials to a Face Set	139
Forcing Katana to Resolve a Material	139
Lighting Your Scene	
Creating a Light in Katana	140

Getting to Grips with the GafferThree Node	142
Gaffer Object Table Overview	143
Creating a Light Using the GafferThree Node	144
Making Use of Rigs	145
Defining a Master Light Material	147
Adding an Aim Constraint to a Light	147
Linking Lights to Specific Objects	148
Disabling a Light for Specific Object	149
Linking Shadows to Specific Objects	149
Adopting Items from Incoming Scene	149
Soloing and Muting Lights and Rigs	150
Locking a Light or Rig's Transform	151
Duplicating an Item Within the Gaffer Object Table	152
Syncing the GafferThree Selection with the Scene Graph	152
Deleting from the Gaffer Object Table	152
Displaying Network Material Parameter Values	152

Creating Shadows	153
Raytracing Shadows in Arnold	153
Raytracing Shadows in PRMan	153
Creating a Shadow Map	154
Creating a Deep Shadow Map	155
Using a Shadow Map in a Light Shader	155

Positioning Lights	156
Moving a Light Within the Viewer	156

Getting to Grips with the Gaffer Node	157
Creating a Light Using the Gaffer Node	157
Making Use of Rigs	157
Defining a Master Light Material	159
Adding a Sky Dome Light	159
Adding an Aim Constraint to a Light	160
Linking Lights to Specific Objects	161
Linking Shadows to Specific Objects	161
Deleting from the Gaffer Object Table	162
Locking a Light or Rig's Transform	162
Soloing and Muting Lights and Rigs	162
Duplicating an Item Within the Gaffer Object Table	163
Syncing the Gaffer Selection with the Scene Graph	163

Rendering Your Scene

Overview	164
Render Types	164
Render Type Availability	167
Influencing a Render	170

Performing a Render	170
Executing Renders with the Render Node	170
Rendering From the Command Line	171
Performing a Preview Render	171

Performing a Live Render	171
Controlling Live Rendering	172
Global Options	173
Renderer Specific Options	176
Setting up Interactive Render Filters	178
Setting up a Render Pass	180
Defining and Overriding a Color Output	180
Defining Outputs Other than Color	182
Defining an AOV Output	182
Previewing Interactive Renders for Outputs Other than Primary	183
OpenEXR Header Metadata	183
Setting up Render Dependencies	184
Managing Color Within Katana	185
Viewing Your Renders	
Using the Monitor Tab	186
Changing the Image Size and Position	187
Overlay Masking	188
Changing How to Trigger a Render	189
Changing the Displayed Channels	190
Changing How the Alpha Channel is Displayed	190
Selecting Which Output Pass to View	191
Using the Catalog tab	192
Viewing the RenderLog for a Catalog Entry	192
Removing Renders from the Catalog	192
Changing the Catalog Renders Displayed in the Monitor tab	193
Manipulating Catalog Entries	193
Changing the Catalog View	195
Using the Histogram	195
Viewing the Pixel Values of the Front and Back Images	196
Comparing Front and Back Images	197
Toggling 2D Manipulator Display	198
Underlaying and Overlaying an Image	198
Rendering a Region of Interest (ROI)	199
Using the Viewer	
Overview	200
Changing How the Scene is Displayed	200
Changing the Layout	200
Changing the Overall Viewer Behavior	202
Using Flush Caches	204

Assigning a Viewer Material Shader	204
Displaying Textures in the Viewer	205
Assigning a Viewer Light Shader	206
Changing Which Lights to Use	206
Changing Shadow Behavior	206
Changing the Anti-Aliasing Settings	206
Changing how Proxies Are Displayed	207
Setting Different Display Properties for Some Locations	207
Changing the Viewer Behavior for Locations that are Selected	209
Changing the Viewer Behavior While Dragging	209
Changing the Background Color	209
Overriding the Display Within a Specific Tab	210
Selecting Within the Viewer	210
Stepping Through the Selection History	210
Changing the View Position	211
Viewport Movement	211
Changing what you Look Through	211
Looking Around the Viewport by Offsetting and Overscanning	213
Changing What is Displayed Within the Viewport	213
Hiding and Unhiding Objects Within the Scene	213
Changing the Subdivision Level of a Subdivision Surface	214
Toggling Grid Display	214
Using Manipulators	214
Toggling Manipulator Display	215
Toggling Annotation Display	217
Toggling the Heads Up Display (HUD)	218
Displaying Normal Information Within the Viewer	218
Transforming an Object in the Viewer	218
Manipulating a Light Source	220
Checking UVs	
Overview	225
Bringing up the UV Viewer Tab	225
Navigating in the UV Viewer Tab	225
Panning	225
Zooming	225
Framing	226
Selecting Faces	226
Adding Textures to the UV Viewer	228
Using Multi-Tile Textures	229
Changing the UV Viewer Display	230

Using the Timeline

Using the Timeline	232

Animating Within Katana

Animation Introduction	234

Setting Keys	236

Toggling Auto Key	236
Setting Keys Manually	236
Baking a Curve	237
Exporting and Importing a Curve	238
Displaying Keyframes	238

Curve Editor Overview	239

Using the Hierarchical View	240
Locking a Curve	240
Hiding and Showing a Curve	241
Switching the Display of a Parameter's Children	241
Setting Keys in the Curve Editor	242
Selecting Keyframes in the Curve Editor	242
Moving Keyframes in the Curve Editor	243
Changing the Display Range in the Curve Editor Graph	243
Changing Display Elements Within the Curve Editor Graph	245
Snapping Keyframes	248
Locking, Unlocking, and Deleting Keyframes	249
Turning a Keyframe into a Breakdown	250
Changing a Segment Function	251
Available Segment Functions	252
Available Extrapolation Functions	255
Changing the Control Points of a Bezier Segment Function	256
Available Tangent Types	257
Baking a Segment of the Curve	260
Smoothing a Segment of the Curve	260
Flipping the Curve Horizontally or Vertically	261
Scaling and Offsetting a Curve	262

Dope Sheet Overview	262

Changing the Displayed Frame Range	263
Selecting Keyframes	264
Moving Keyframes	264
Creating a Keyframe from an Interpolated Value	265
Copy and Pasting Keyframes	265
Deleting Keyframes	266
Toggling Tooltip Display	266

Groups, Macros, & Super Tools

Groups	267

Macros	267
Adding User Parameters	268
Conditional Behavior	270
Super Tools	273

LiveGroups

Creating a LiveGroup	275
Editing LiveGroup Parameters	276
Loading and Reloading a LiveGroup	276
Editing the Contents of a LiveGroup	277
Making a LiveGroup Node Editable	277
Modified State of Editable LiveGroup Nodes	278
Publishing a LiveGroup	279
LiveGroup Conversion	280

User Parameters and Widget Types

Parameter Types	281
Widget Types	283
Widget Type Availability	291

Collections and CEL

Overview	293
CEL in the User Interface	293
Guidelines for using CEL	294

Working with Attributes

Overview	297
OpScript Nodes	298
Adding an OpScript	298
OpScript Tutorials	300
Creating Scene Graph Locations	300
Deleting Scene Graph Locations	301
Copying Scene Graph Locations and Attributes	302
AttributeSet and AttributeScript Nodes	303
OpScript vs AttributeScript	303

Making Changes with the AttributeSet Node	304
Adding an AttributeScript	305
Using Python within an AttributeScript Node	306
Understanding Locations	306
Understanding Inheritance	307
Using Initialization Scripts	307
When to Run	308
Getting Multiple Time Samples	308
Custom Error Messages with an AttributeScript Node	308
Example Python Scripts	313
Arbitrary Attributes Within Katana	318
Beyond the AttributeSet and AttributeScript Nodes	319
The FnAttribute Module	319
Accessing the FnAttribute Module	320
Working with Data Attributes	321
Working with Group Attributes	324
Invalid Attributes (C++ only)	326
Error Checking	327
Debugging	328
Look Files	
Overview	329
Handing off Looks from Look Development to Lighting	329
Look File Baking	330
Other Uses of Look Files	330
How Look Files Work	331
Setting Material Overrides using Look Files	332
Collections using Look Files	332
Look Files for Palettes of Materials	333
Look File Globals	333
Lights and Constraints in Look Files	333
The Look File Manager	334
Look Development with Look Files	
Look File Overview	335
Using Look Files to Create a Material Palette	335
Creating a Material Palette	335

Reading in a Material Palette	336
Using Look Files in an Asset's Look Development	336
Creating a Look File Using LookFileBake	336
Assigning a Look File to an Asset	342
Resolving Look Files	342
Overriding Look File Material Attributes	343
Activating Look File Lights and Constraints	343
Using Look Files as Default Settings	344
Making Look Files Easier with the LookFileManager	345
Bringing a Look File into the Scene Graph	345
Removing a Look File from the Look Files List	347
Managing Passes in the LookFileManager	347
Overriding Look Files	348
 Scene Data	
Scene Attributes and Hierarchy	350
Common Attributes	350
The Process of Generating Scene Graph Data	352
Structured Scene Graph Data	353
Bounding Boxes and Good Data for Renderers	353
Proxies and Good Data for Users	354
Level-of-Detail Groups	355
Alembic and Other Input Data Formats	356
ScenegraphXML	356
 Graph State Variables	
Setting Graph State Variables	358
Global Variables	358
Local Variables	359
Inspecting Graph State Variables	359
Reading Graph State Variables	360
Nodes	360
Scripts	361
How Do Graph State Variables Work?	361
 Resolvers	
Overview	362
Examples of Resolvers	363

Implicit Resolvers	363
Creating your own resolvers	364

Handling Textures

Introduction	366
Texture Handling Options	366
Materials with Explicit Textures	366
Using Material Overrides to Specify Textures	366
Using the {attr:xxx} Syntax for Shader Parameters	367
Using Primvars in RenderMan	369
Using User Custom Data	369
Using Pipeline Data to Set Textures	370
Metadata on Imported Geometry	370
Metadata Read in from Another Source	370
Processes to Procedurally Resolve Textures	370

Asset Management

Asset Plug-ins	371
The Asset Publishing Process	372
Choosing an Asset Plug-in	372
Example Asset Plug-in	373
Retrieve and Publish	374
Retrieving	375
Publishing	376

Instancing

Overview	378
Benefits of Instancing	378
Instancing in PRMan	379
Instance Types	379
Per-Instance PrimVars	379
PRMan Example	380
Instancing in Arnold	383
Arnold Example	383

Appendix A: Keyboard Shortcuts

Conventions	387
Customizing Keyboard Shortcuts	387

Keyboard Shortcut Manager	388
General Keyboard Shortcuts	389
Backdrop Node Keyboard Shortcuts	390
Curve Editor Keyboard Shortcuts	390
Dope Sheet Keyboard Shortcuts	391
Messages Keyboard Shortcuts	391
Monitor Keyboard Shortcuts	392
Node Graph Keyboard Shortcuts	393
Parameters Keyboard Shortcuts	395
Python Console Keyboard Shortcuts	396
Timeline Keyboard Shortcuts	396
Viewer Keyboard Shortcuts	397

Appendix B: Expressions

Expressions	400
Variables Within Expressions	400
Katana Expression Functions	401
Python Modules Within Expressions	403

Appendix C: CEL and Collections

Collection Expression Language (CEL)	406
Basic CEL Syntax	406
Value Expressions	407
CEL Sets	407
Collections	408
Collection Syntax	408
CEL Pattern Extensions	408
Example CEL Pattern Extensions	409

Appendix D: AttributeScript to OpScript Conversion

Functions	410
GetAttr	410

SetAttr	412
GetName, GetFullName	413
XForm Manipulation	413
Util.AssetResolve	413
User Parameters	413
Language Basics	414
Caveats	417

Appendix E: External Software

	419
Third-Party Library Versions	419
Third-Party Library Licenses	420
GNU Lesser General Public License (LGPL) v2.1	485
GNU Lesser General Public License (LGPL) v3.0	493
GNU General Public License (GPL) v3.0	496
OpenSceneGraph v0.0	506

Appendix F: External File Formats

Args Files	509
Args Files In Shaders	509
Edit Shader Interface Interactively in the UI	510
Widget Types	510
Widget Options	514
Conditional Visibility Options	514
Conditional Locking Options	515
Editing Help Text	515
Grouping Parameters into Pages	515
Co-Shaders	516
Co-Shader Pairing	516
Example of an Args File	517
Args Files for Renderer Procedurals	518
UI Hints for Plug-ins Using Argument Templates	518
AttributeFiles	521
Overview	521
Current Limitations	521
Usage in Katana	521
Example Scene Using AttributeFile	522
Example of a Simple XML AttributeFile	522
ScenegraphXML	523
Overview	523
Using ScenegraphXML in Katana	524
Reading and Writing ScenegraphXML from Python	524

Writing ScenegraphXML data from Maya	525
Alembic Maya Plug-Ins	529
Example Maya Export Scripts	529
ScenegraphXML attributes in Maya	530
Custom Attributes in Alembic Components	530
Data Format Description	530
ScenegraphXML.py Classes and Methods	533

Appendix G: End User License Agreement

End User License Agreement (EULA)	536

Preface

Katana is a 3D application specifically designed for the needs of look development and lighting in an asset-based pipeline. Originally developed at Sony Pictures Imageworks, Katana has been their core tool for look development and lighting for all their productions since *Spider-Man 3*, *Beowulf*, and *Surf's Up!*.

Katana provides a very general framework for efficient look development and lighting, with the goals of scalability, flexibility, and supporting an asset-based pipeline.

Key Concepts

A recipe in Katana is an arrangement of instructions - in the form of connected nodes - to read, process, and manipulate a 3D scene or image data. A Katana project can be made up of any number of recipes, and development of these recipes revolves around two tabs: the **Node Graph** and **Scene Graph** tabs.

Within the **Node Graph** tab, Katana utilizes a node-based workflow, where you connect a series of nodes to read, process, and manipulate 3D scene or image data. These connections form a non-destructive **recipe** for processing data. A node's parameters can be viewed and edited in the **Parameters** tab.

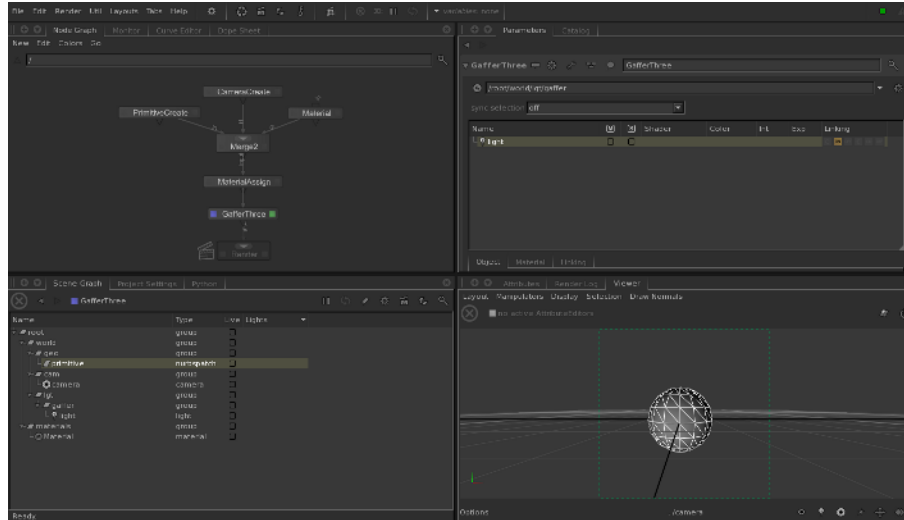
To view the scene generated up to any node within a recipe, you use the **Scene Graph** tab. The scene graph's hierarchical structure is made up of locations that can be referenced by their path, such as **/root**. Each location has a number of attributes which represent the data at that location. You can view, but not edit, the attributes at a location within the **Attributes** tab.

These key concepts are explained in greater depth in the chapter [The Way of The Katana](#).

An Example Recipe

In this example of a very basic recipe:

- the **Node Graph** tab contains the recipe for creating the scene,
- the **Scene Graph** tab shows the scene generated at the **beauty** node (a renamed Render node),
- the **Parameters** tab shows the current parameters of the GafferThree node,
- the **Viewer** tab shows a 3D view from the point of view of the camera.



User Interface and Naming Conventions

For clarity, some naming conventions have been adopted throughout this User Guide.

User interface (UI) elements are in **bold**, such as the **Node Graph** tab and **cameraName** parameter.

Panes are user interface areas that contain one or more tabs. For instance, when you first open Katana there are four panes displaying four tabs. In the top left pane are the **Node Graph**, **Monitor**, **Curve Editor**, and **Dope Sheet** tabs.

As mentioned briefly above, the **Scene Graph** tab displays the scene data generated up to a certain node. Sometimes, the data displayed is mentioned without the UI being relevant; when this is the case, the term scene graph (without the bold) is used.

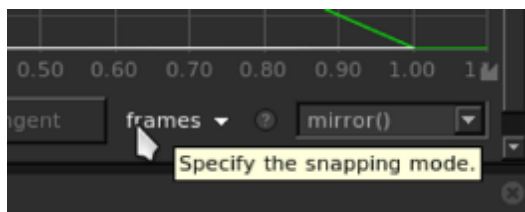
Getting Help

If you can't find what you need in this document, there are other sources of help available to you for all aspects of Katana and its operation.

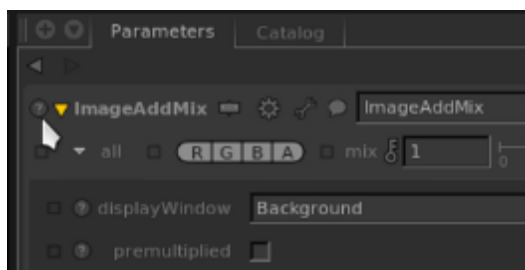
Viewing Additional Help

Katana features several forms of help:

- Some controls offer concise instructions in the form of tooltips. To display the tooltips, move your mouse pointer over a control or node parameter.



- Many nodes include contextual descriptions of the node's parameters when viewed in the **Parameters** tab. To display these descriptions, click the ? icon.



- Finally, you can click the **Help** menu to access the following:
 - **User Guide** - the Katana user guide, which is aimed at users of all levels and covers most operations inside Katana.
 - **Technical Guide** - a more technical overview of Katana, aimed at those with a more technical role such as pipeline engineers.
 - **Reference Guide** - a reference list of the nodes, their parameters, and how to use them.
 - **Documentation** - a full list of all the accompanying documents and examples.
 - **Keyboard Shortcuts** - a full list of all Katana shortcuts.
 - **API Reference** - information on Katana APIs.
 - **Example Projects** - a list of accompanying example files.
 - **I want a pony** - this command accesses one to the most important core features of Katana, which has been vital part of delivering many productions. Whenever you realize that a pony is required, this feature is there to help you.

Contacting Customer Support

Should questions arise that the guides or in-application help system fail to address, you can contact Customer Support directly by e-mail at support@thefoundry.co.uk.

Please note that technical support is only provided during UK hours.

Sending Crash Reports

You can set a specific e-mail address to send crash reports to using the following environment variable:

KATANA_CRASH_REPORT_EMAIL

For example, to send crash reports to the address `jsmith@example.com`, set the environment variable to:

```
KATANA_CRASH_REPORT_EMAIL=jsmith@example.com
```

You can set a specific crash reporting mode using the `KATANA_CRASH_REPORT_EMAIL_MODE` environment variable with the following values:

- **0** - NO -- No prompt, no e-mail.
- **1** - YES -- No prompt, send e-mail.
- **2** - PROMPT -- Ask the user whether to send e-mail.

The crash report is also saved in your **/tmp** directory using the following format:

```
/tmp/katana_crash_dump_yyyymmdd_hh_mm_ss.txt
```

Installation and Licensing

Installing and licensing new applications can be a boring task that you just want to be done with as soon as possible. To help you with that, this chapter guides you to the point where you have a default Katana workspace in front of you and are ready to start.

System Requirements

Before you do anything else, ensure that your system meets the following minimum requirements to run Katana effectively:

- Katana 2.1v3 is tested and qualified on Linux 64-bit CentOS/RHEL 5.4
- A graphics card which supports OpenGL shader model 4.
- A supported renderer (see [Supported Renderers](#) on the next page)



NOTE: Due to Python's handling of imports on case-insensitive platforms (see PEP 235), it is not possible to run Katana from a file system location on a network-attached storage device (NAS) that has been set up with mount options for case-insensitive names.

Third-Party Dependencies

Katana version 2.1v3 has dependencies on the following third-party libraries:

- OpenEXR 2.0.1
- OpenSSL 1.0.0.a

These libraries are provided in the Katana distribution, in separate directories under `${KATANA_HOME}/bin`

An ABI-compatible copy of these libraries needs to reside on your `LD_LIBRARY_PATH` in order for many of Katana's plug-ins to run. The Katana application itself uses `RPATHs` to locate the required libraries.



NOTE: Katana's wrapper script `${KATANA_HOME}/katana` appends `${LD_LIBRARY_PATH}` to ensure these libraries are visible to Katana plug-ins.

If you manage your own `LD_LIBRARY_PATH` or wish to expose these libraries to plug-ins by some other means, you can call the Katana binary directly using:
`${KATANA_ROOT}/bin/katanaBin`

Supported Renderers

Katana 2.1v3 supports PRMan 17.0, Arnold 4.2.4.0, and Arnold 4.1.3.3. The supplied renderer plug-ins are compiled, and tested against these versions, using GCC 4.1.2. Minor version increments of PRMan and Arnold may work, as long as they are API compliant with the supported versions.

Support for PRMan 18.0 and above are provided directly by Pixar using their RenderMan Studio for Katana plug-in. To get access to RenderMan Studio for Katana please contact Pixar directly.



NOTE: To use Katana with PRMan, or Arnold, you must install a compatible version of RenderMan ProServer, or Arnold. Katana does not ship with a renderer.

To use a version of PRMan or Arnold other than those listed above, you may need to recompile the renderer plug-in.

To expose new features and portions, you may need to modify the renderer plug-in.



NOTE: Currently, it is not possible to load more than one plug-in version of the same renderer. For example, it's not possible to load the plug-in for PRMan 17.0 and PRMan 18 at the same time. However, it is possible to load PRMan 17.0 and Arnold 4.1.3.3.

Using a version of PRMan or Arnold other than those listed above may produce unexpected behavior. Using a compiler other than GCC 4.1.2 may produce unexpected behavior. Please note that we can only guarantee to respond to Katana bug reports when they are reproducible with the supplied versions of the renderer plug-in, compiled with the supported version of GCC.

Install Katana

The current version of Katana can be obtained from our website:

<http://www.thefoundry.co.uk/products/katana/downloads>

Once you have downloaded the tarball, follow the installation instructions below:

1. Move the tarball into a temporary folder.
2. Extract and decompress the tarball inside the temporary folder.
tar xvf Katana<version>-linux-x86-release-64.tgz
3. Start the install script:
./install.sh
4. After reading the End User License Agreement (EULA), if you agree with it, type:
yes

5. Enter the installation directory for Katana.
6. That's it! Proceed to [Launching Katana](#) below.



TIP: You can also use the **--path** option which assumes you have read, and agree with, the EULA. For instance, to use the **--path** option to install Katana to the `/opt/foundry/katana` directory, execute the install script with:

```
./install.sh --path /opt/foundry/katana
```

KATANA_RESOURCES

Katana uses the `KATANA_RESOURCES` environment variable to provide a list of paths under which to look for plug-ins and other customisations (such as shelves, tabs and resolutions).

Examples are provided in the following directory, and are loaded if this path is included in the `KATANA_RESOURCES` environment variable:

```
$KATANA_ROOT/plugins/Resources/Examples
```

Connecting Katana to a Renderer

Katana is not tied to any one renderer, in fact, it is designed to be renderer agnostic. By using the Renderer API, plug-in developers connect renderers and renderer-specific settings to Katana. Included with Katana are plug-ins for Solid Angle's Arnold and Pixar's Photorealistic RenderMan (PRMan).

Before trying to connect Katana to a renderer, make sure the renderer is installed correctly. Consult the manual that accompanies the renderer for details.



NOTE: For more information on writing a renderer plug-in for Katana utilizing the Rendering API, see the developers documentation that accompanies the installation. The developers documentation can be accessed through the **Help > Documentation** menu option inside Katana.

Katana uses the `KATANA_RESOURCES` environment variable to find the renderer plug-ins it needs. The renderer plug-ins included with Katana reside in the **plugins/Resources** directory inside the installation location. Currently, the renderer plug-ins included are:

- Arnold v4.1
- Arnold v4.2
- PRMan v17.0

To use the Arnold v4.2 and PRMan v17.0 plug-ins, set the environment variable to:

```
KATANA_RESOURCES=$KATANA_ROOT/plugins/Resources/Arnold4.2:$KATANA_ROOT/plugins/Resources/PRMan17
```



NOTE: This assumes the `KATANA_ROOT` environment variable has been previously set to the current installation location.

Changing the Default Renderer

The default renderer is specified using the `DEFAULT_RENDERER` environment variable. For example, if you're using a bash shell:

```
export DEFAULT_RENDERER=prman
```

This default is used by nodes and tabs that require renderer-specific information in instances where the renderer is not specified by the recipe at the currently viewed node. If this environment variable is not set, **prman** is used by default.



NOTE: If the requested renderer plug-in is not available, Katana displays warning messages, and in certain cases, error messages.

Arnold Specific Notes

The current version of the Arnold plug-in may not be compiled against the exact same version of the renderer that your studio uses. To this end, Katana includes the source code for the plug-in so you can match it to your current version.

Compiling the Arnold Plug-in to Match Your Arnold Version

Included in `${KATANA_ROOT}/plugins/Src/ArnoldX.X/src` are source directories, so studios can compile the Arnold renderer plug-in to match the version of Arnold they use. Makefiles are included within both directories (RendererInfo and RendererPlugin) to make this compilation easier.

PRMan Specific Notes

Katana currently supports PRMan v17.0.

Including the Katana PRMan Shaders

Katana includes a number of basic PRMan shaders. Although not production optimized, some studios may find them useful as example code. They are located in **`${KATANA_ROOT}/plugins/Resources/PRMan17/Shaders`**. Katana's PRMan plug-in finds shaders through the `RMAN_SHADERPATH` environment variable. To include the example shaders, append their location to the environment variable. For instance:

```
RMAN_SHADERPATH=$RMAN_SHADERPATH:${KATANA_ROOT}/plugins/Resources/PRMan17/Shaders
```

Network Configuration

When performing a Live Render, or Preview Render, Katana uses TCP/IP sockets for communication between the render process and **Monitor** tab. Therefore, Katana needs to be able to resolve the workstation host name by means of a DNS. For this to work ensure one of the following:

- Your corporate network has a valid DNS server running, capable of resolving the host name of your machine.
- Add an entry to your **/etc/hosts** which explicitly maps your host name to IP address.

Python Search Path

Katana's Python module search path is configured as follows, leaving the PYTHONPATH environment variable unchanged for child processes:

```
KATANA_INTERNAL_PYTHONPATH =
    KATANA_DEBUG_PRE_PYTHONPATH :
    <Katana internal Python paths> :
    PYTHONPATH and site customizations :
    KATANA_POST_PYTHONPATH
```

This allows Katana to initialize its environment safely, avoiding inadvertent loading of unsupported modules. You may add to the search path using Python **site** customizations or by setting the KATANA_POST_PYTHONPATH environment variable.

Additionally, the KATANA_DEBUG_PRE_PYTHONPATH environment is provided, for debugging purposes only, as it may lead to unexpected application behaviour due to non-supported modules loading in place of the application's.



NOTE: Changes to **sys.path** included in **sitecustomize.py** do not affect Katana internal Python paths.

Launching Katana

To start Katana in the default, Interactive mode:

1. Open a terminal.
2. Navigate to the directory where you installed Katana.
3. Enter:
./katana

If a license is present, the interface displays. Otherwise, you need to license Katana. See [Licensing Katana](#).

You can also specify a Katana scene to load. To start in Interactive mode, and open a specified Katana scene:

1. Open a terminal.
2. Navigate to the directory where you installed Katana.
3. Enter:

```
./katana /yourDirectory/yourScene.katana
```

You can also start Katana in a number of different modes:

- Interactive mode is the default mode. It requires no additional command line arguments, and as the only launch mode that starts Katana with the GUI.
- Batch mode opens a Katana scene for render farm rendering.
- Shell mode exposes Katana's Python interpreter in the terminal shell.
- Script mode runs a specified Python script in Katana's Python interpreter.

For information on starting Katana in the other launch modes, see the *Launch Modes* chapter in the *Katana Technical Guide*.

Licensing Katana

About Licenses

To use Katana, you need a valid **floating license** and a server running the Foundry Licensing Tools (FLT). Katana uses RLM licensing, and the default local RLM location is:

```
/usr/local/foundry/RLM
```

Floating Licenses - also known as **counted** licenses - enable one of our products to work on any networked client machine. The floating license should only be installed on the server machine and is locked to a unique number on that server. Floating licenses often declare a port number on the server line. You also need to install the Foundry License Tools (FLT) on the server machine to manage the floating licenses and hand them out to the client machines. The FLT is freely available to download from our website.

Setting up the Floating License Server

All the tools necessary for setting up a license server are included with the Foundry Licensing Tools (FLT). The latest version can be downloaded at

<http://www.thefoundry.co.uk/support/licensing/tools>

You should use the Foundry License Utility (FLU) to install the floating license on the server machine. When you install the floating license on the server, the FLU provides information on how to point client machines to your server.

The FLU is included with the FLT install but you can also download a copy from <http://www.thefoundry.co.uk/support/licensing/tools>



NOTE: Although Katana is only available for Linux, you can install the license server software on Mac, Windows, or Linux. See the license server system requirements for its own supported operating systems.

Setting up the License on the Client Machine

Once the license server is up and running, you must point your client machines towards the license server using the details given when the floating license was installed. You can either use the Foundry License Utility (FLU) to create a client license or you can manually set the environment variable `foundry_LICENSE` to point to the server. The correct syntax for the environment variable is `<PORT_ID>@<SERVER_NAME>`. For instance: `foundry_LICENSE=4101@our_license_server`.

Further Information

Please see our website for more information about licensing:

<http://www.thefoundry.co.uk/support/licensing/>

Configuring Message Level

Katana uses the standard Python logging module and, by default, logs messages of types info, warning, error, and critical. Messages shown in the UI are generated by the root logger, which is configured with the `${KATANA_ROOT}/bin/python_log.conf` file. To change the level of message generated, edit the `logger_root level` parameter in `python_log.conf` to one of the options listed below:

- **DEBUG**

Generates messages of debug level and higher.

- **INFO**

Generates messages of info level and higher.

- **WARNING**

Generates messages of warning level and higher.

- **ERROR**

Generates messages of error level and higher.

- **CRITICAL**

Generates critical messages only.

Whether messages of a particular - generated - type are displayed or not is controlled through the **Message Window** options. For more on viewing error messages and notifications, see [The Message Center](#).

Defining the directory in which Katana stores temporary files

When launching Katana, a temporary folder is created in the standard directory of temporary files and folders. This is used to store data, such as caches for instance, for the current session.

The name of the Katana temporary directory uses the following pattern: **katana_tmpdir_[process-ID]**

By default, the Katana temporary directory is created in the folder as returned by the **tempfile.gettempdir()** function from the **tempfile** module in the Python Standard Library. You can override the directory by setting the **TMPDIR** environment variable to a specific file system location.

You can obtain the exact file system location that Katana uses through the **KATANA_TMPDIR** environment variable, which, when interrogated from within a running Katana session, may contain value like the following: **/tmp/katana_tmpdir_26458**



NOTE: The Katana temporary directory is removed once the Katana session terminates.

The Way of The Katana

Katana Introduction

For users of 3D applications and compositors, Katana has familiar elements, such as a timeline, a Node Graph, a hierarchical scene view, and a 3D viewer. Whether you are a 3D veteran or diving into your first 3D application, this chapter explains how some of these elements are used within Katana and tries to get you into the mindset of a Katana user.

Unlike most of the chapters, this one guides you through a number of steps and then follows up with an explanation of how these steps influence Katana behind the scenes. If you don't understand everything at first glance, don't panic, it's all explained in greater detail in later chapters. In fact, there are cross-references to more in-depth explanations dotted throughout this chapter.

The steps in this chapter are:

[The Way of The Katana](#)

[Step 1: Learning About the Node Graph, Recipes, and Node Creation](#)

[Step 2: Editing a Node and Using the Parameters Tab](#)

[Step 3: Creating and Assigning Materials](#)

[Step 4: Lights, Camera, Action](#)

[Step 5: Using the Scene Graph](#)

[Step 6: Using the Viewer](#)

[Step 7: Starting a Render](#)

It's time to launch the application and get ready to embrace *The Way of the Katana!*

Making a Scene

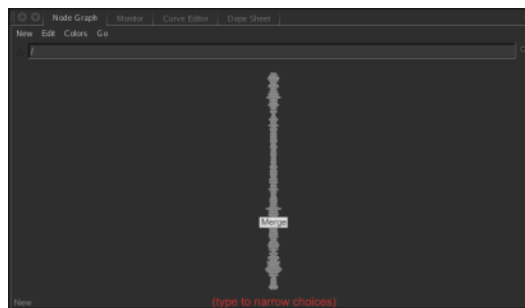
The primary user interface in Katana is the **Node Graph** tab, where you add nodes to your recipes in order to create, import, or manipulate objects and shaders. Nodes have editable parameters, and are interconnected, passing data from node-to-node along the connections. The flow of data is one-way (downstream), with the output of one node passed as an input to its downstream neighbor (or neighbors).

The node graph itself is not what Katana sends to the renderer. Instead, Katana constructs a scene graph of scene data from the contents and interconnections of the node graph, and it's this scene graph data that is sent to your renderer.

Step 1: Learning About the Node Graph, Recipes, and Node Creation

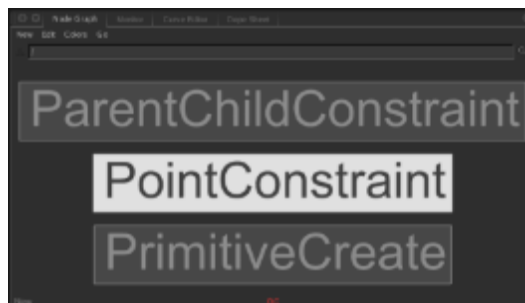
1. Hover the mouse over the **Node Graph** tab and press **Tab**.

All available nodes are displayed.



2. Type **PC**.

This narrows the node list down to those with **PC** at the start of their name and those with **PC** as their name's starting capital letters.

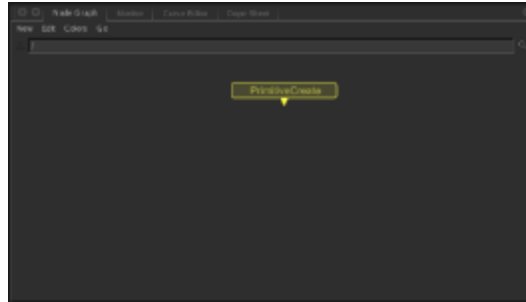


3. Click **PrimitiveCreate**.

The PrimitiveCreate node is selected. Once selected, it floats with the cursor, ready to be placed.

4. Click somewhere towards the top of the **Node Graph**.

The node is added to the **Node Graph**, beginning a new recipe.



The **Node Graph** is where it all starts. It is here that you create a recipe by connecting various nodes, which add, override, or modify scene data.

Most recipes start by reading in the 3D elements - such as camera data, geometry caches, or particle caches - that comprise the scene. In this example, we use a PrimitiveCreate node that defaults to a sphere.

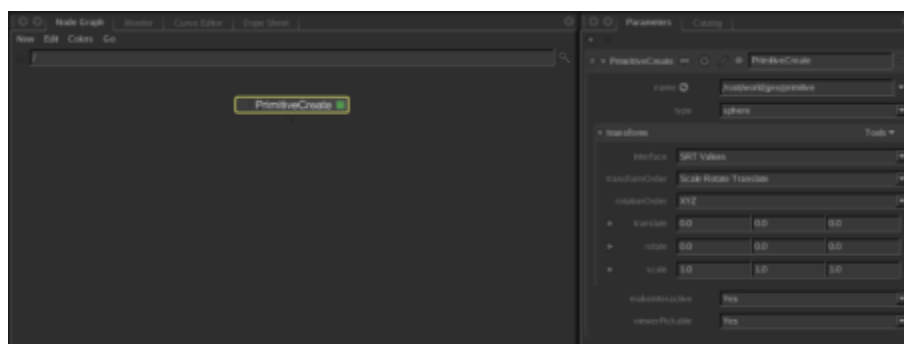
One of the most important things to understand about the **Node Graph**, and its resulting recipe, is that it is non-destructive. The recipe is a description of how to bring in the ingredients (the various assets), modify them to suit the shot, add materials and assign them to objects, add lights, and finally send everything off to a renderer. The recipe approach is extremely flexible, allowing the assets to be continually updated in an iterative workflow.

For more on nodes and the Node Graph, see [Working with Nodes](#).

Step 2: Editing a Node and Using the Parameters Tab

Hover the mouse over the PrimitiveCreate node and press **E**.

A green square displays at the right-hand side of the node and the node's parameters are displayed in the **Parameters** tab.



This is one way to make a node editable. You can also:

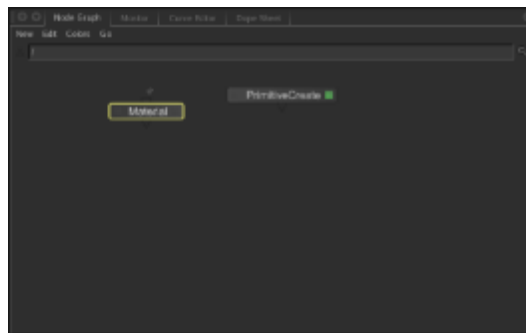
- select one or more nodes and press **Alt+E**, or
- click the faint square at the right-hand side of the node.

Most nodes within Katana have **parameters** that modify their behavior. You can change these parameters in the **Parameters** tab. A node that is being edited within the **Parameters** tab displays a green square on its right-hand side.

To find out more about parameters and the **Parameters** tab, see [Editing a Node's Parameters](#).

Step 3: Creating and Assigning Materials

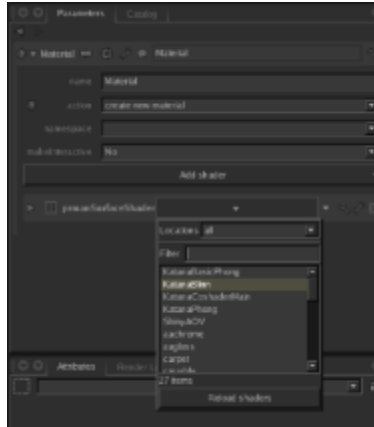
1. Press **Tab** in the **Node Graph**.
2. Type **MAT** to filter the node list.
3. Select **Material**.
4. Click to the left of the PrimitiveCreate node to add the Material node to the **Node Graph**.



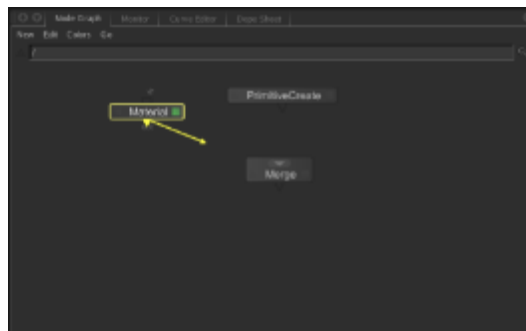
5. Hover the mouse over the Material node and press **E**.
The Material node becomes editable within the **Parameters** tab.
6. In the **Parameters** tab, click **Add shader** and select a shader type from the list, for instance **prman surface**.
The shader list varies depending on the renderers installed.



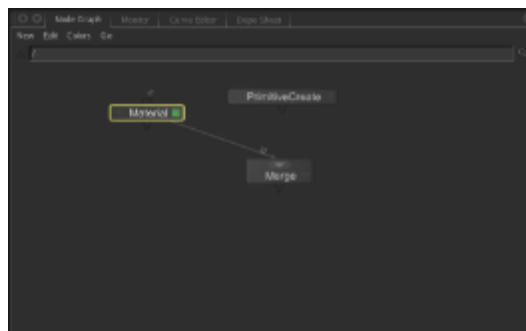
7. Click the large dropdown to the immediate right of the shader type and select a shader, for instance **KatanaBlinn**.



8. Create a Merge node and place it below the PrimitiveCreate node.
9. Hover the mouse over the Material node and press ' (**Backtick**).
A connection from the Material node is started.



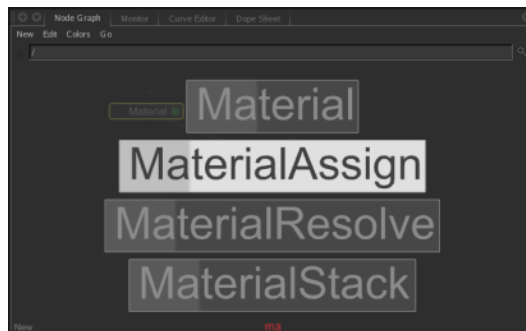
10. Hover the mouse over the Merge node and press '**(Backtick)**'. Pressing **Backtick** a second time connects the Material node to the input of the Merge node. It is also possible to manually connect two nodes by dragging from the output triangles to the input squares, see [Connecting Nodes](#).



11. Connect the PrimitiveCreate node to the Merge node, as described above.
The **Merge** node brings different branches of the recipe together, combining their scene data. Right now, the Merge node brings together the sphere created by the PrimitiveCreate node and the material created by the Material node.



12. Create a MaterialAssign node using the **Tab** key menu.

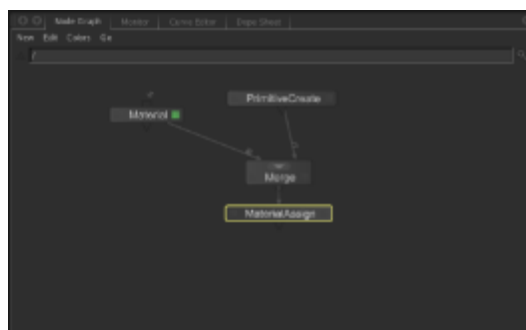


13. With the MaterialAssign node floating with the cursor, hover the node over the output from the Merge node until it turns yellow, then click to connect.

This connects the MaterialAssign node to the output of the Merge node. The MaterialAssign node continues to float with the cursor.



14. Click below the Merge node to place the MaterialAssign node.

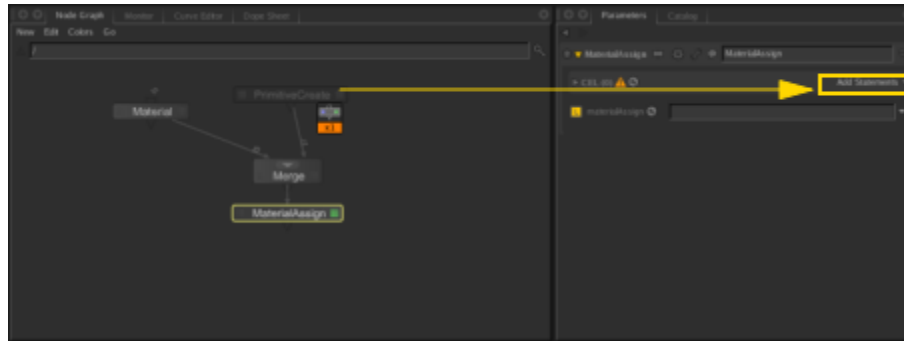


15. Hover the mouse over the MaterialAssign node and press **E**.

The MaterialAssign node becomes editable within the **Parameters** tab.

16. **Shift**+middle-click and drag from the PrimitiveCreate node to the **Add Statements** menu in the **Parameters** tab.

The scene graph location of the object created by the PrimitiveCreate node is added to the **CEL** parameter.

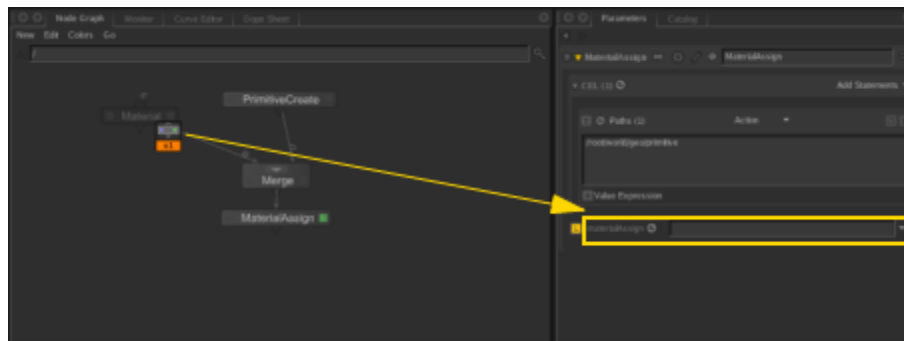


After you've created a material it needs to be assigned. In the MaterialAssign node, Katana uses the **Collection Expression Language (CEL)** to create a list of scene graph locations to be used in the material's assignment.

For a more comprehensive explanation, and an explanation of locations, see [Using the Scene Graph](#).

For further details on CEL, see [Assigning Locations to a CEL Parameter](#).

17. **Shift**+middle-click and drag from the Material node to the **materialAssign** parameter in the **Parameters** tab.
An expression is created that links the material created by the Material node to the **materialAssign** parameter. If the location created by the Material node changes, the expression automatically updates the **materialAssign** parameter with the new location.

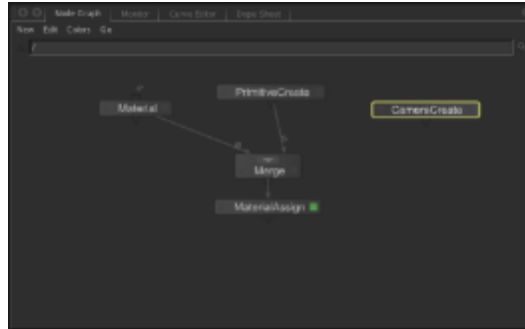


Materials define how geometry and lights are rendered. Each material can have one or more shaders for each renderer, as well as a shader that defines how that object is displayed in the 3D Viewer. Materials can be assigned to geometry or lights and then saved as a **Katana Look File (.klf)**. This part of a production pipeline is commonly referred to as **look development**.

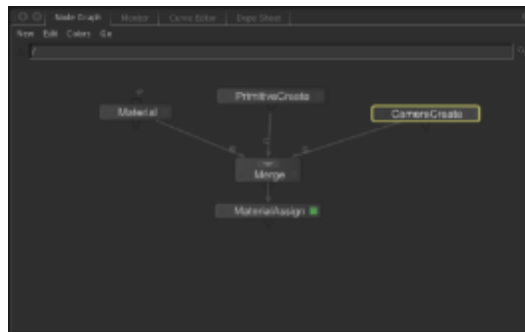
Katana Look Files are extremely powerful. For a more in-depth explanation, see [Look Development with Look Files](#).

Step 4: Lights, Camera, Action

1. Create a CameraCreate node in the same way as the previous nodes and place it to the right of the PrimitiveCreate node.



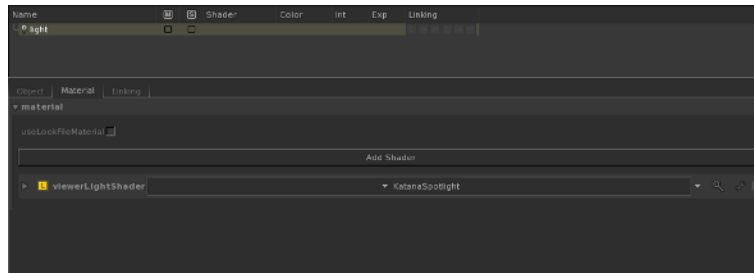
2. Connect the CameraCreate node to the Merge node.



3. Create a GafferThree node and connect it to the output of the MaterialAssign node.



4. Hover the mouse over the GafferThree node and press **E**.
The GafferThree node becomes editable within the **Parameters** tab.
5. Right-click on the gaffer object table in the **Parameters** tab and select **Add > Light**.
This creates a light and places it in the gaffer object table.
6. In the gaffer object table, under the **Shader** column, right-click, select **Add Shader**, and choose **light** from the dropdown menu. Click on the **Material** sub-tab and select a light from the list, for instance **KatanaSpotlight**, for the **viewerLightShader** dropdown.



NOTE: The shader list varies depending on the renderers installed.

In the example, the camera is created within the recipe. It could just as easily be brought in from an external file, such as Alembic (ABC). Supplementary cameras may be placed in Katana for various rendering techniques, such as camera based projections or stereo.

Lights are usually created within the GafferThree node. The name comes from the role of the person on-set responsible for the setting up of the lights - the Gaffer. The GafferThree node provides a one-stop-shop for a number of convenient lighting functions, for instance: light creation, shader assignment, and light soloing and muting.

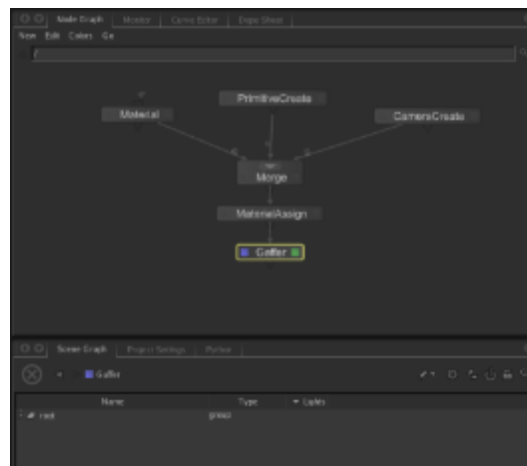
For more information on the GafferThree node, see [Getting to Grips with the GafferThree Node](#), or for lighting in general, see [Lighting Your Scene](#).

Step 5: Using the Scene Graph

1. Hover the mouse over the GafferThree node and press **V**.

The **Scene Graph** tab now displays the 3D scene generated up to the GafferThree node.

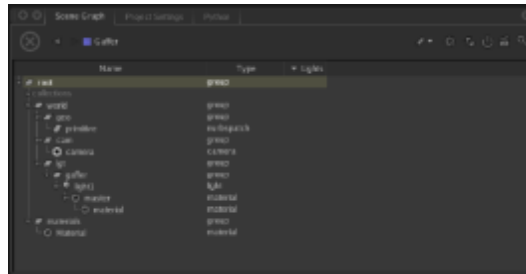
This is one way to view the scene graph generated at a node. You can also click the faint square at the left-hand side of the node.



When the **Scene Graph** tab is displaying the scene generated at a node, that node (referred to as the view node) has a blue square displayed on the left-hand side.

2. In the **Scene Graph** tab, right-click on the **/root** location and select **Expand All**.

The **/root** location expands to display everything within the scene graph.



There are a few key concepts regarding the scene graph:

The first key concept is that: **the Scene Graph is just a viewer**. The scene graph displays the 3D scene generated for the current frame by stepping through the recipe up to the node with the blue square - this node is the current view node.

The second concept is that: **there is no such thing as the scene in Katana**. You can merge and branch the recipe, pruning and adding to one of the branches. Therefore, pressing **V** at different nodes can generate vastly different scenes. To see this for yourself, hover the mouse over the CameraCreate node and press **V**. The scene graph changes because at this node in the recipe, only the camera has been created. You can view the 3D scene generated at any of the other nodes in the same way. When you are happy, hover the mouse back over the GafferThree node and press **V** again.

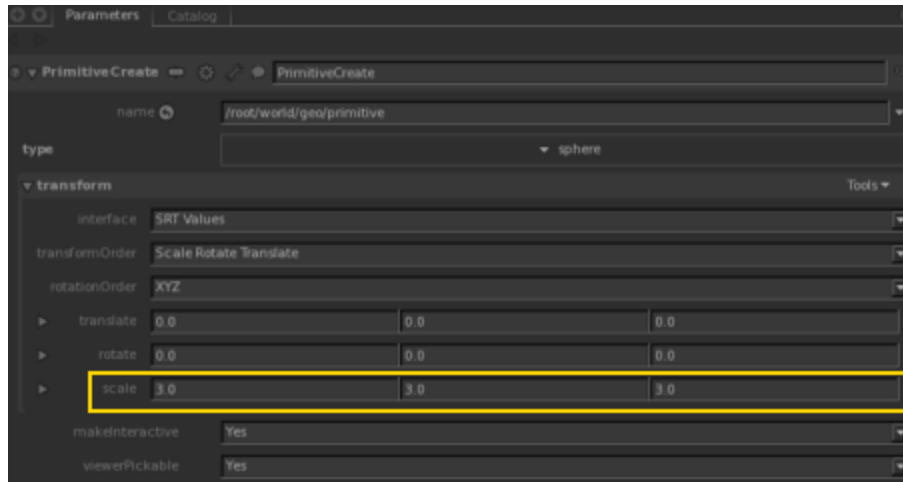
Lastly, **the Scene Graph only loads data as it is needed**. This deferred loading makes it possible for Katana to contain recipes dealing with scenes of incredible and potentially even infinite size. As you control which elements are loaded - by expanding locations within the scene graph - you can light extremely complicated scenes by only loading the required data. You don't need to load all the scene data to light the scene.

Each location, such as **/root** or **/root/world**, within the scene graph has attributes that you can view within the read-only **Attributes** tab. To find out more about the scene graph, locations, and attributes, see [Using the Scene Graph](#).

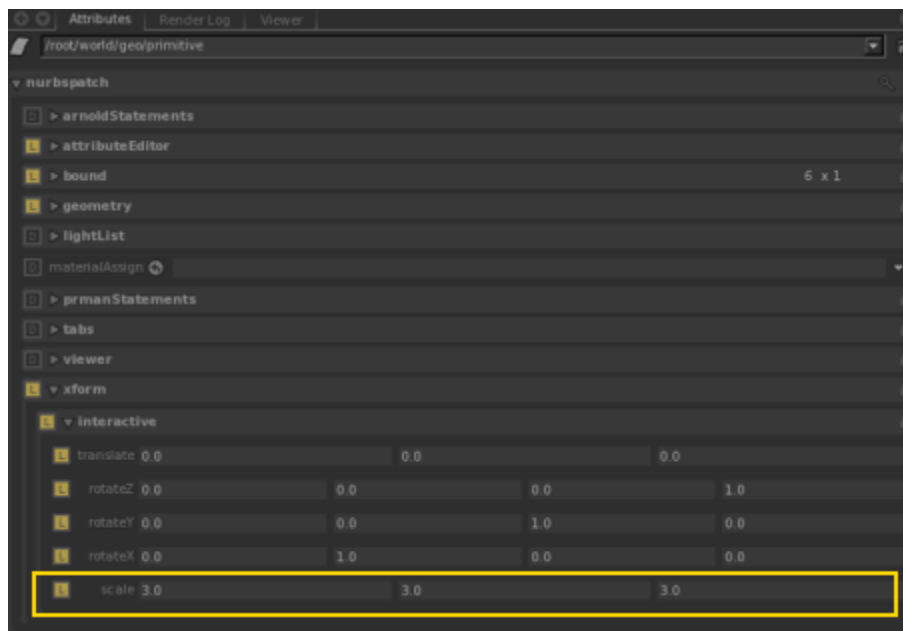
There are three main viewers for the underlying scene graph data in Katana, the **Scene Graph** tab, the **Attributes** tab, and the **Viewer** tab.

Locations in the scene graph have attributes, which are determined by parameters on nodes in the node graph. For example:

1. Add a PrimitiveCreate node to an empty Katana project.
2. Edit the node's parameters so that it's a type sphere, scaled to 3.0 in each of the X, Y, Z axes.



Katana creates a sphere primitive in the scene graph from the information in the PrimitiveCreate node in the node graph. Select the primitive location in the **Scene Graph** tab and look at its **Attributes** tab.

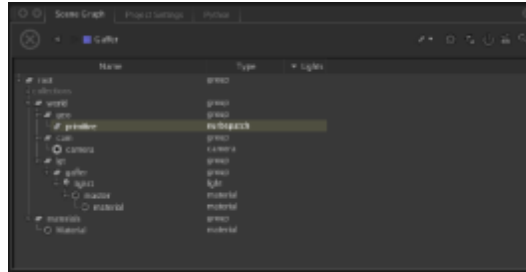


3. Under xform.interactive the scale is set to 3.0 in each of the X, Y, Z axes. Go back to the PrimitiveCreate node in the Node Graph and enter a Z scale value of 5.0.

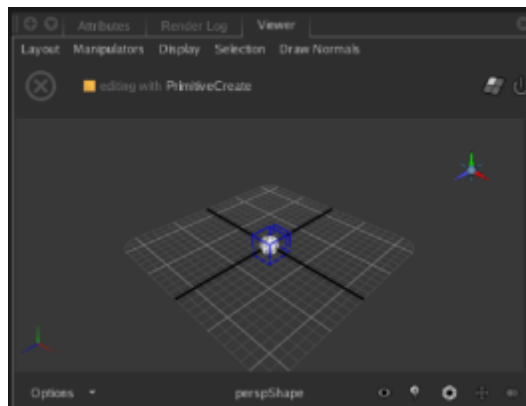
Now go back to the primitive location in the scene graph and see that under xform.interactive, the Z scale has changed to 5.0.

Step 6: Using the Viewer

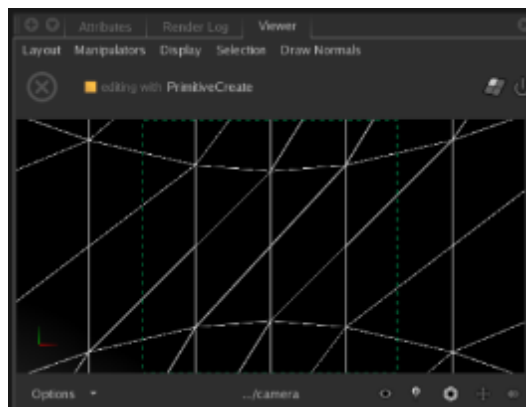
1. With the GafferThree node as the view node (designated by the blue square) and the scene graph fully expanded, select **primitive** in the **Scene Graph** tab.



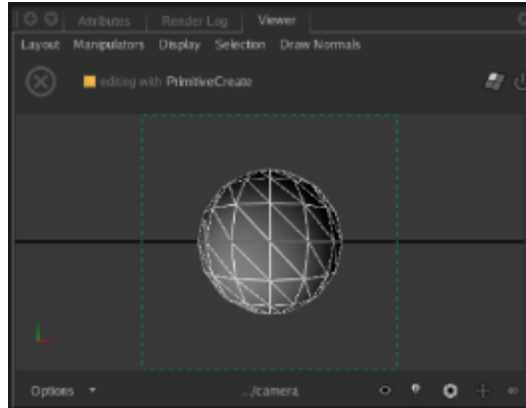
2. Towards the bottom of the **Viewer** tab (located in the bottom right pane by default), click **perspShape**.
A list of objects you can look through displays. This is a combination of the light list (to list just lights, click ) and the camera list (to list just cameras, click ).



3. Select **../camera**.
The **Viewer** tab now shows the view from the point of view of the camera.

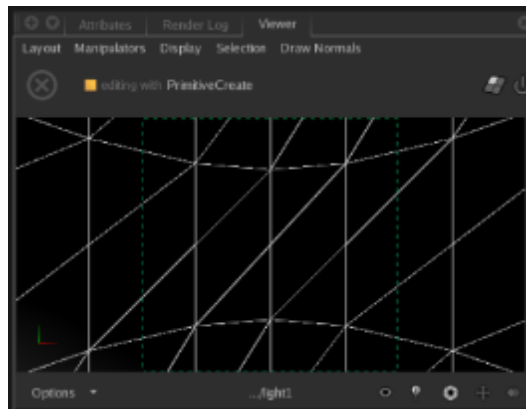


4. With **primitive** still selected in the **Scene Graph** tab, press **F** in the **Viewer** tab.
The camera moves to frame the currently selected object and makes it the camera's point of interest.

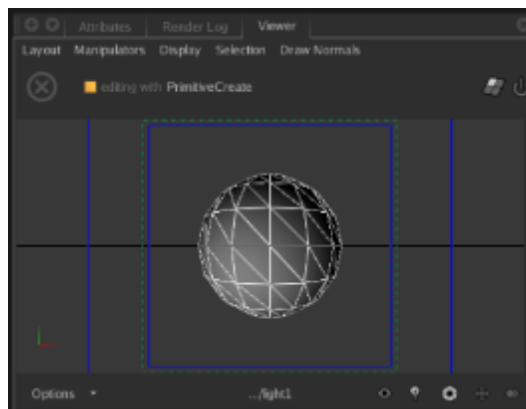


5. Click **../camera** in the **Viewer** tab and select **../light**.

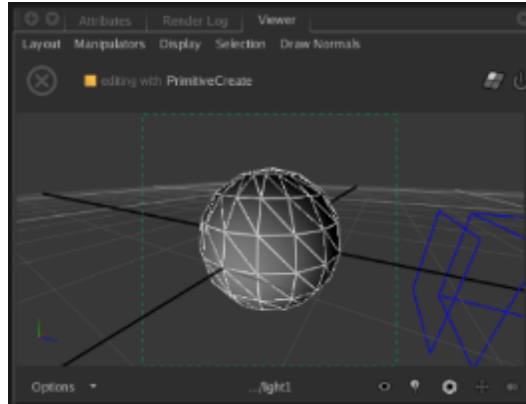
The view changes to that of the light.



6. Press **F** in the **Viewer** tab to frame the sphere again, this time for the light.



7. **Alt**+left-click and drag to rotate the light around the sphere and position the light.



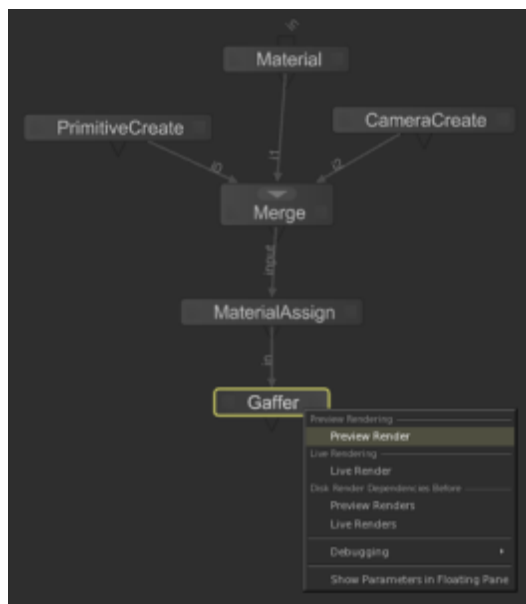
The Viewer is a 3D representation of the scene within the scene graph but only locations exposed within the Scene Graph are displayed. Therefore, if you first view the scene graph for a node and only **/root** is exposed the Viewer is empty.

To learn more about how to move around and manipulate objects within the **Viewer**, see [Using the Viewer](#).

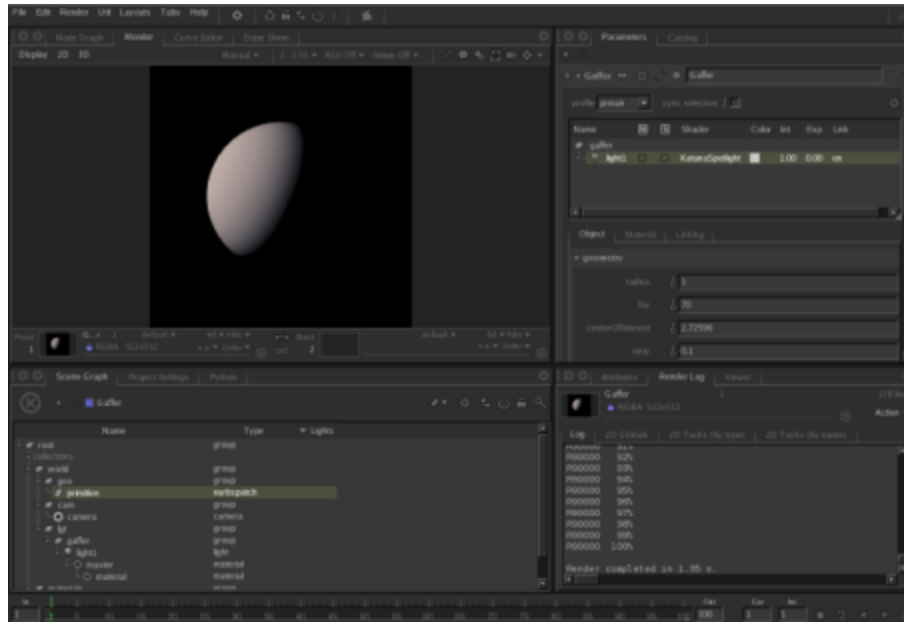
Step 7: Starting a Render

Katana has a number of render options, and their availability depends on the type of node you try to render from. For the render options available at each type of node, see the [Rendering Your Scene](#) chapter.

To perform a preview render of the scene created at [Making a Scene](#), right-click on the GafferThree node, and select **Preview Renders**. The scene generated at the GafferThree node is sent to the renderer. This uses your production renderer, not an internal Katana renderer (Katana does not have an internal renderer of its own).



You can view the progress of the render in the **Render Log** tab and the results are displayed in the **Monitor** tab (click the **Monitor** tab next to the **Node Graph** tab when using the default workspace to access the **Monitor**). To have the render fit the size of the **Monitor** tab, press **F**.



For more on rendering, saving your renders, and changing render settings, see [Rendering Your Scene](#). For more on viewing the results of your renders, see [Viewing Your Renders](#).

That's it! You now know the basics on wielding Katana. Keep going!

Customizing Your Workspace

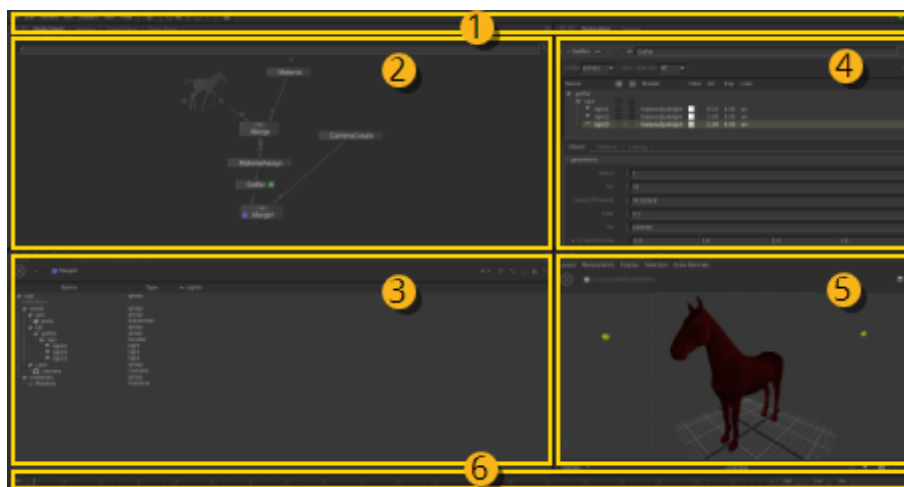
This chapter is designed to help you understand the Katana workspace, and how to customize it to meet your particular needs.

Workspace Overview

If you have used 3D applications in the past, you may notice that Katana's workspace has many familiar features, such as a timeline, a hierarchical **Scene Graph** tab, an OpenGL viewer, and a 2D monitor.

The Default Workspace

Here is an illustration of a simple Katana workspace.



1. The menu bar, complete with menus, such as **File** and **Help**, and menu icons, such as the Interactive Render Filter icon, and the Messages menu. For further details, see [Menu Bar Components](#).
2. The top left pane, containing the **Node Graph**, **Monitor**, **Curve Editor**, and **Dope Sheet** tabs.
3. The bottom left pane, containing the **Scene Graph**, **Project Settings**, and **Python** tabs.
4. The top right pane, containing the **Parameters** and **Catalog** tabs.
5. The bottom right pane, containing the **Attributes**, **Render Log**, and **Viewer** tabs.
For more on the contents of the various tabs, see the [The Default Tabs](#) below.
6. The **Timeline**. The **Timeline** is explained in greater depth in [Using the Timeline](#).

The Default Tabs

The following are the tabs displayed by default. More tabs are available in the **Tabs** menu.

Tab	Function
Node Graph	This is where you build your node tree (a tree graph that represents the recipe for manipulating a 3D scene).
Monitor	This is where you view the results of your renders and composites.
Curve Editor	Lets you edit animation keys as curves.
Dope Sheet	Lets you edit animation keys as a spreadsheet of keys and ranges.
Scene Graph	This is where you view the scene data, generated at the current view node in the Node Graph, in a hierarchical representation. The objects - such as geometry, particle data, volumetric data, materials, cameras, and lights - that make up the scene graph are called locations, and are referenced by their path, such as /root/world/cam/camera .
Project Settings	This is where you can view and edit parameters for the whole project.
Python	This is where you can enter Python commands as well as view their outputs. It acts as a Python interactive shell within Katana.
Parameters	This is where you adjust the parameters associated with nodes currently selected for editing.
Catalog	Lets you view and organize previous renders.
Attributes	Lets you view the attribute values held at each location in the scene graph.
Render Log	Lets you view text output from the renderer.
Viewer	This is where you can view and manipulate your scene using a 3D representation. Only objects whose locations that are visible in the Scene Graph tab are displayed.

Menu Bar Components

The Katana menu bar includes the following functions:

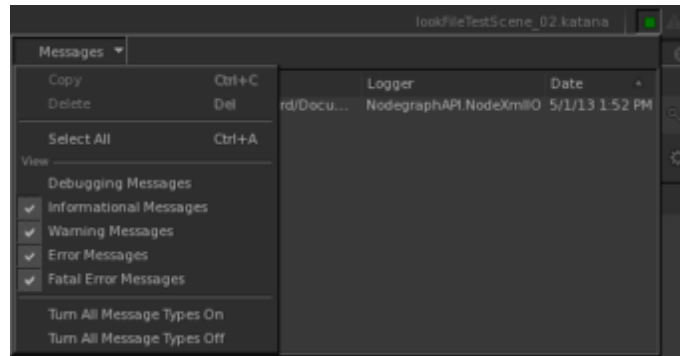
Menu	Functions
File	Commands for disk operations, including creating, loading, and saving Katana projects.
Edit	Undo, redo, and preferences.
Render	Rendering the output.
Util	A group of miscellaneous menu items including farm management and cache handling.
Layouts	Adjusting, saving, activating, and deleting layouts.
Tabs	Adding floating panes to the interface.
Help	Accessing documentation, APIs, and information on the current version.
	Collection of Python shelf scripts.
	Flush caches: forces assets, such as look files, to be dropped from memory and reloaded when needed.
	Toggles implicit resolvers. This gives a better impression of the data sent to the renderer at the cost of extra computation. For more on implicit resolvers, see Turning on Implicit Resolvers .
	When enabled, rendering only includes items selected in the Scene Graph tab.
	The auto key icon: when enabled, changing parameters automatically adds a new key.
	Specify what interactive render filters to use for any new interactive renders. For more on interactive render filters, see Setting up Interactive Render Filters .
	Cancels a live render that is currently in progress.

Menu	Functions
	Specifies whether live rendering is set to update: - manually, - when changes to materials, lights, or geometry transformations are made or a parameter change is applied (Pen-up), - continuously when changes are made to materials, lights, or geometry transformations, including some manipulations in the Viewer tab (Continuous).
	When live rendering is set to Manual, this button triggers an update. This button is not clickable for either Pen-up or Continuous modes.
	Displays any graph state variables that have been set in the Project Settings tab. If no variables have been set, the icon says variables: none; if there are variables set, these are displayed in yellow and can be changed dynamically to influence those in the Project Settings tab, or the other way around. For more information on how to set variables, refer to Setting Graph State Variables in the Graph State Variables chapter.

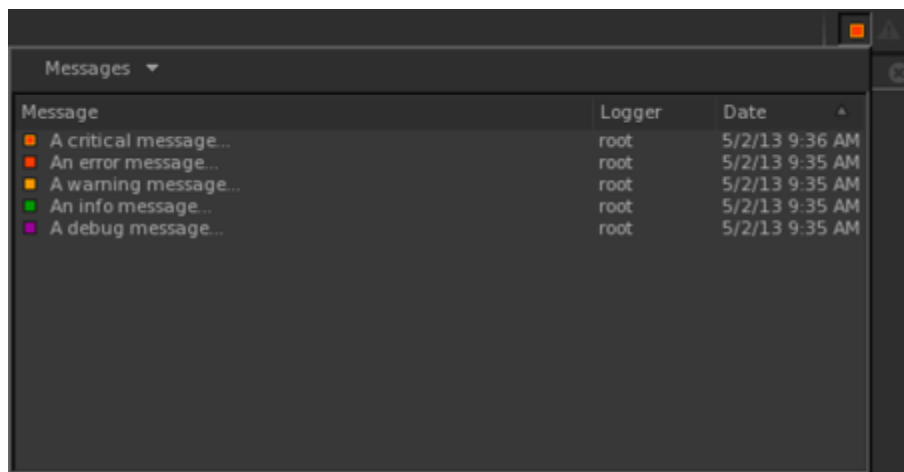
The Message Center

Katana records messages on activities such as loading scenes, and converting scripts between different render versions. The message menu is at the right-hand edge of the menu bar, and operates on a traffic light system, with Informational Messages marked green, Warning Messages in orange, and any errors in red. If you click on the message menu icon the messages window opens. The message menu icon itself changes color to match that of the most serious message in the list (so can be either green, orange, or red).

By default, the messages window displays Informational Messages, Warning Messages, Error Messages, and Fatal Error Messages. Opening the message window, then clicking on **Messages** opens the message window options dropdown menu, where you can enable or disable display of messages by category, copy selected messages, or delete selected messages.



The message window shows a truncated summary of each message. If you select, then copy a message, you copy the full text, which you can paste into a text editor.



Customizing Your Workspace

You can create layouts designed for whatever function you happen to be performing. For instance: lighting, look development, or material editing. You can then save your preferred layouts for future use.

During the customization process, you can:

- Resize panes to create space where it's most needed.
- Maximize the pane under the mouse cursor.
- Move and split panes to create new work areas, for example, to have two **Viewers** side-by-side.
- Remove panes and all tabs nested inside them.
- Add and remove tabs as required.
- Move tabs to easily access the elements you often need.
- Float and nest tabs to create more space or group similar functions together in the same pane.
- Add a **Timebar** to the main Katana window or any tab.
- Make the main Katana window fullscreen, hiding the window borders.

Once you are happy with the layout, you can save it for future use.



NOTE: The **Layouts** menu also includes four preset layouts to get you going.

Adjusting Layouts

To make accessing the elements you often need as quick and easy as possible, it's a good idea to adjust the default layout(s).

Resizing Panes

To resize individual panes, hover the mouse over the divider line until the cursor changes to the resize icon. Click and drag the cursor to resize the pane.



TIP: When moving the divider line, by default, if it crosses multiple panes, the entire line is moved. To only move the divider line for the local pane, **Ctrl**+drag.

Maximizing Panes

To maximize a pane so that it expands to the size of the window:

- Click  in the top left corner of the pane, or
- Hover over the pane and press **Spacebar**, or
- Double-click the tab of the pane to maximize.

To return to the regular interface, click  or press **Spacebar**.



NOTE: Pressing the **Spacebar** in the **Monitor** tab does not maximize the pane, instead it swaps the **Front** and **Back** images.

Moving and Splitting Panes

To move an existing pane to a new location in the interface:

1. Hover over the  icon in the top left corner of the pane until the cursor changes to the move icon.
2. Click and drag the pane to a new location.

The orange highlight around the destination pane helps you determine where the pane is moved and whether the destination pane is split horizontally or vertically.

Removing Panes

To remove a pane and all tabs nested inside it, right-click on any of the tab names and select **Close all**.

Adding Panes

To add a floating pane to the interface, use the **Tabs** menu in the menu bar and select the tab you want to add.

To add a tab to a specific pane, click  in the top left corner of the pane and select the tab you want to add.

Moving Tabs

To move an existing tab to a new location in the interface, click and drag the tab to a new location.

The orange highlight around the destination pane helps you determine where the tab is nested and if the destination pane is split horizontally or vertically.

Removing Individual Tabs

- Make sure you are viewing the tab you want to remove and click on  in the top right corner of the pane, or
- Right-click on the name of the tab and select **Close tab**.

Floating and Nesting Tabs

To turn a tab into a floating window, right-click on the name of the tab and select **Detach tab**.

To nest a floating tab, click on the name of the tab and drag it to where you want it to dock. Use the orange highlight around the destination pane to help you determine where the tab is nested and whether the destination pane splits horizontally or vertically.

Showing and Hiding Timelines

To show or hide a timeline at the bottom of the main Katana window, select **Layouts > Show Main Timeline**.

To show or hide a timeline at the bottom of any tab, right-click on the tab name and select **Show Timeline**.

Making the Main Window Fullscreen

To make the main Katana window fullscreen, select **Layouts > View Fullscreen**.

To return to normal, select **Layouts > View Fullscreen**.

Saving Layouts

You can save as many of your favorite layouts as needed, retrieving them as necessary.

To save a layout:

1. Once you are happy with your layout, select **Layouts > Save Current Layout**.
The **Save Current Layout** dialog opens.
2. In the dialog, enter a name for the new layout.
3. If your layout includes any floating tabs and you want those to be saved with the layout, check **Save # Floating Panes** (where # corresponds to the current number of floating panes).
4. Click **Save** to preserve your layout.

Loading Layouts

To load a previously saved layout, select it from the **Layouts** menu in the menu bar.

Deleting Layouts

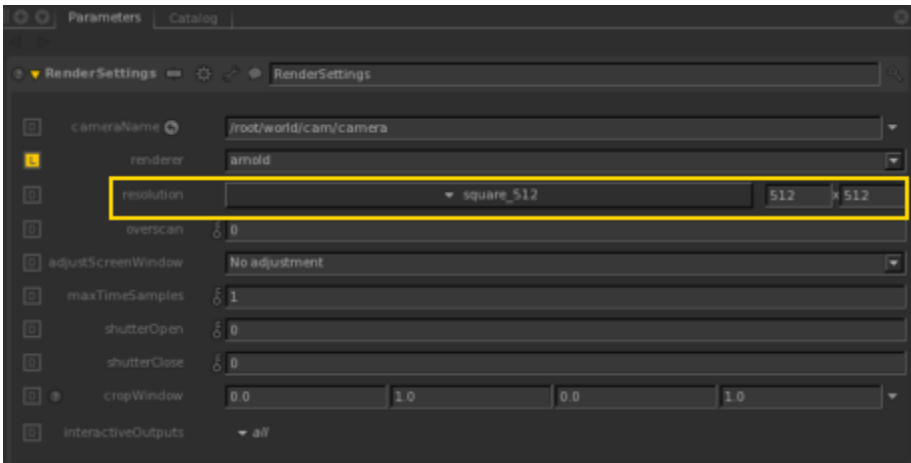
1. Select **Layouts > Edit Saved Layouts**.
2. In the dialog that opens, select the layout to delete from the list available.
3. Click **Delete Layout** and **Save**.

Custom Render Resolutions

You can define custom render resolutions to supplement or replace Katana's pre-defined resolutions. You can define resolutions through the UI, using Python, or with XML files. For more information on defining resolutions using Python, or XML, see the *Custom Render Resolutions* chapter in the *Katana Technical Guide*.

Using the UI

You can set the render resolution through any node with a **resolution** field, such as a **RenderSettings** or **ImageColor** node. Each node with a **resolution** field has a dropdown menu of pre-defined resolutions, and text entry boxes for manual definition.



Resolutions defined manually are saved - and recalled - with the Katana project, but are not saved for use in other Katana projects. If you select a pre-determined resolution, that selection is saved - and recalled - with the Katana project.



NOTE: The **resolution** field in the **Tab > Project Settings** window specifies the resolution for 2D image nodes, not 3D renders.

For more on setting custom render resolutions, see the *Custom Render Resolutions* chapter in the *Katana Technical Guide*.

Preferences

This section shows you which preferences you can set up for Katana.



NOTE: To reset the preferences to their default values, you can delete the **Prefs2.ini** file in the following location: `$KATANA_HOME/katana`.

Application

Control (UI)	Default Value	Function
Application		

Control (UI)	Default Value	Function
fontFamily	Bitstream Vera Sans	Set the font family for the Katana UI. Some UI components cache elements like font or control size. Restart Katana for the font family change to take full effect.
fontSize	8	Set the base font size for the Katana UI. Some UI components cache elements like font or control size. Restart Katana for the font size change to take full effect.
macroSavePath	N/A	Specify the default location for saving macros.
crashFile		
numberOfActions	50	Set the number of actions before automatically saving the current project to a file from which the project can be restored after a crash.
time	5	Set the time in minutes before automatically saving the current project to a file from which the project can be restored after a crash.
rendering		
interactiveRenderThreadOverride	Yes	Use the interactiveRenderThreads settings below to control the number of threads used in an interactive render. When set to No , the renderer-specific settings from the scene are used.
interactiveRenderThreads2D	9	Set the number of render threads for interactive 2D renders, including hot renders. This value has to be larger than 0 .
interactiveRenderThreads3D	1	Set the number of render threads for interactive 3D renders, including hot renders.
restartLiveRenderOnTimeChange	Yes	Specify whether or not a Live Render session should be automatically restarted when the selected time changes.

Control (UI)	Default Value	Function
saveKatanaScriptOnRender	No	Specify whether or not to auto-save the .katana script for each render.  WARNING: For large sessions, this can substantially increase render launch times.
updateMode2D	Manual	Specify what UI actions trigger a render of the currently viewed 2D node. • Manual - changes to materials, lights, or geometry transformations don't trigger a render update. To have the changes take effect, click the Trigger 2D Update button. • Pen-up - changes to materials, lights, or geometry transformations trigger a render update only when the mouse button is released or a parameter change is applied. • Continuous - changes to materials, lights, or geometry transformations, including some manipulations in the Viewer tab, continuously trigger a render update.  NOTE: Setting this preference to a value other than Continuous is particularly useful when rendering large images where constant render updates would be too time-consuming.

Control (UI)	Default Value	Function
updateMode3D	Continuous	Specify what UI actions trigger an update (recook) of the scene. • Manual - changes to materials, lights, or geometry transformations do not trigger an update of the scene. To have the changes take effect, click the Trigger 3D Update button . • Pen-up - changes to materials, lights, or geometry transformations trigger an update of the scene only when the mouse button is released or a parameter change is applied. • Continuous - changes to materials, lights, or geometry transformations, including some manipulations in the Viewer tab, continuously trigger an update of the scene.

Color

Control (UI)	Default Value	Function
color		
useSingleComponentRGB	No	Set whether or not the RGB gradients are displayed using pure components.

Dopesheet

Control (UI)	Default Value	Function
dopesheet		
showToolTips	Yes	Specify whether or not to display the tooltips with key values.

ExternalTools

Control (UI)	Default Value	Function
externalTools		
editor	gedit	Enter the command for launching an external text editor.
imageViewer	eog	Enter the command for launching an external image viewer.
pdfViewer	evince	Enter the command for launching an external PDF viewer.
webBrowser	firefox	Enter the command for launching an external web browser.

Layout

Control (UI)	Default Value	Function
layout		
default	N/A	Specify which Katana UI layout you want to start in by default.
hotkey1	Monitor	Specify the layout assigned to F9 .
hotkey2	Lighting	Specify the layout assigned to F10 .
hotkey3	Dual Lighting	Specify the layout assigned to F11 .
hotkey4	Composite	Specify the layout assigned to F12 .

Monitor

Control (UI)	Default Value	Function
monitor		
allowKeylessTabletInteraction	Yes	Specify whether or not to allow the tablet interaction without pressing the modifier keys. When set to Yes, you can use the stylus pen buttons to perform actions such as zooming, panning, and so on. When set to No, you need to use modifier keys along with the stylus pen to perform actions.
renderIDPass	Yes	Specify whether or not an ID pass is rendered when rendering from a node in the node graph. The ID pass allows the Pixel Probe in the Monitor tab to identify pieces of geometry under the pointer, for which the name of a corresponding scene graph location is then displayed in the Pixel Probe toolbar.  TIP: You can also toggle on and off the Render ID Pass preference in Katana's main menu in the Render > 3D Rendering > Render ID Pass menu, or in the Monitor tab in the 3D > Render ID Pass menu.
showColorTimingEditor	No	Specify whether or not to display the color timing editor.
showCommentEditor	No	Specify whether or not to display the comment editor.
showOverlayControl	No	Specify whether or not to display the overlay controls.
showPanel	Yes	Specify whether or not to display the monitor panel at each render.
showPixelProbe	No	Specify whether or not to display the pixel probe.
showSwipeLine	Yes	Specify whether or not to display the swipe line.

Control (UI)	Default Value	Function
ShowTextResolution	Yes	Specify whether or not to display the resolution name for the image sizes in the monitor information.
backgroundColor	N/A	Choose the color values. For more information, refer to the Color Widget Type in the <i>Reference Guide</i> , in the <i>Common Parameter Widgets</i> chapter.
manipulatorLockedColor	N/A	
manipulatorNormalColor	N/A	
manipulatorSelectColor	N/A	
swipeLineColor	N/A	

NodeTags

Control (UI)	Default Value	Function
nodeTags		
menuCategories	constraint,lookfile, resolve,output,color, composite,source, i/o,keying,transform,filter,foundry,views, _macro	Enter the categories to display in the node menu.

Nodegraph

Control (UI)	Default Value	Function
nodegraph		

Control (UI)	Default Value	Function
allowKeylessTabletInteraction	Yes	Specify whether or not to allow the tablet interaction without pressing the modifier keys. When set to Yes, you can use the stylus pen buttons to perform actions such as zooming, panning, and so on. When set to No, you need to use modifier keys along with the stylus pen to perform actions.
autoConnectOnCreate	No	Specify whether or not to auto-connect new nodes based on selection.
autoScroll	Yes	Specify whether or not to auto-scroll while selecting, dragging, and so on.
dimNodesUnconnectedToViewedNode	No	Specify whether or not to dim nodes unconnected to the viewed node.
dimNonContributingNodes	No	Specify whether or not to dim nodes not contributing to the viewed node.
drawLowContrast	Yes	Specify whether or not to draw the node graph with a low-contrast look.
findOnlyNodesInThisGroupDefault	Yes	Specify whether or not to set the default state for the Show Only Nodes in this Group checkbox in the find pop-up of the Node Graph tab.
flagErrorsFileIn	Yes	Specify whether or not to flag FileIn node errors automatically.
flagErrorsNodeConnection	Yes	Specify whether or not to flag node connection errors automatically.
lockStickyNoteNodes	No	Specify whether or not to lock backdrop nodes.
showExpressionLinks	No	Specify whether or not to display expression links.
showNodeIcons	Yes	Specify whether or not to display icons on nodes where available.

Control (UI)	Default Value	Function
showOffscreenFlagArrows	Yes	Specify whether or not to display off-screen flag arrows.
showRolloverNodeNames	Yes	Specify whether or not to display node names on rollover.
showViewMasks	No	Specify whether or not to display the view mask flags and link colors.
snapToGrid	No	Specify whether or not to snap nodes to grid.
stickyDrag	No	When this preference is set to Yes , nodes that are dragged in the Node Graph tab stick to the pointer when the mouse button is released, so that they can be moved without keeping the mouse button held down. In this mode, nodes are placed only when the mouse button is clicked again. This grab-and-drop behavior may help prevent repetitive strain injury (RSI).

Nodes

Control (UI)	Default Value	Function
nodes		
nodes > gaffer		
syncSelection	off	Specify the default selection sync option for the Gaffer.

Parameters

Control (UI)	Default Value	Function
parameters		

Control (UI)	Default Value	Function
cacheEditors	Yes	Specify whether or not to cache node parameters editors.
openCelParametersEditors	No	Specify whether or not to open CEL parameters editors.
selectParameterValueOnFirstClick	No	Specify whether or not the text of parameter values (including project settings and preferences) is selected when first clicking into a value field.  NOTE: By default, when clicking into a value field, the text cursor is placed where the text has been clicked, and no text is selected. This single-click-to-select behavior may help prevent repetitive strain injury (RSI).

Python

Control (UI)	Default Value	Function
python		
autoCompletionBehavior	Shell	Select the auto-completion behavior in the Python tab. Choose from these two options: • Shell - when pressing the Tab key, tokens are completed based on matches with possible completions. If there is more than one possibility for completion, a list of possibilities is printed to the result area of the tab. • IDE - while typing in the command area of the tab, possible completions are shown in a pop-up widget. Select completions with the up and down arrow keys, and chosen by pressing the Tab or Return/Enter key. Press Esc to close the pop-up widget.

Control (UI)	Default Value	Function
showHelpTooltips	No	Specify whether or not to display tooltips with help on the item under the pointer within the Python tab, both in the result and the command areas of the tab.

Scenegraph

Control (UI)	Default Value	Function
scenegraph		
showLightsColumn	Yes	Choose whether or not to display the Lights column in the Scene Graph tab.
scenegraph > userColumns		
userColumns > Add	N/A	Select from the dropdown menu and define the custom columns to display in the Scene Graph tab.

Viewer

Control (UI)	Default Value	Function
viewer		
interactiveProcessingDelay	250	The Viewer delays the processing of interactive updates for a short period of time. This allows updates to be processed in batches, and increases responsiveness. This setting determines the length of this delay in milliseconds.
showImagePlanes	Yes	Specify whether or not to display all image planes.
showOverlayControl	No	Specify whether or not to display the overlay controls.
showPanZoomEditor	No	Specify whether or not to display the pan/zoom controls.

Creating a Katana Project

Katana Projects and Recipes

There are no fixed rules as to what constitutes a Katana project. A Katana project is simply a collection of recipes that are worked on together and stored in a single **.katana** file. A project could be a shot, a scene, or look development for one or more assets.

Each recipe within a project can be totally self-contained or it can be linked to others through dependencies. As an example, look development could have one recipe which creates a **Katana look file (.klf)** for a piece of geometry and another recipe which renders out a turntable of that same geometry complete with its newly created Katana look file assigned.

How you group your recipes into Katana projects is up to you and your studio.

Creating a New Katana Project

To create a new Katana project:

1. Select **File > New** (or press **Ctrl+N**).
2. If needed, click **New Project** in the **Unsaved Changes** dialog window to confirm.



NOTE: **Ctrl+N** does not work within the **Node Graph**.

Saving a Katana Project

To save your current Katana project:

Select **File > Save** (or press **Ctrl+S**).

If the file has not been saved before, the **file browser** displays. See steps 2 to 4 below to select a location to save.

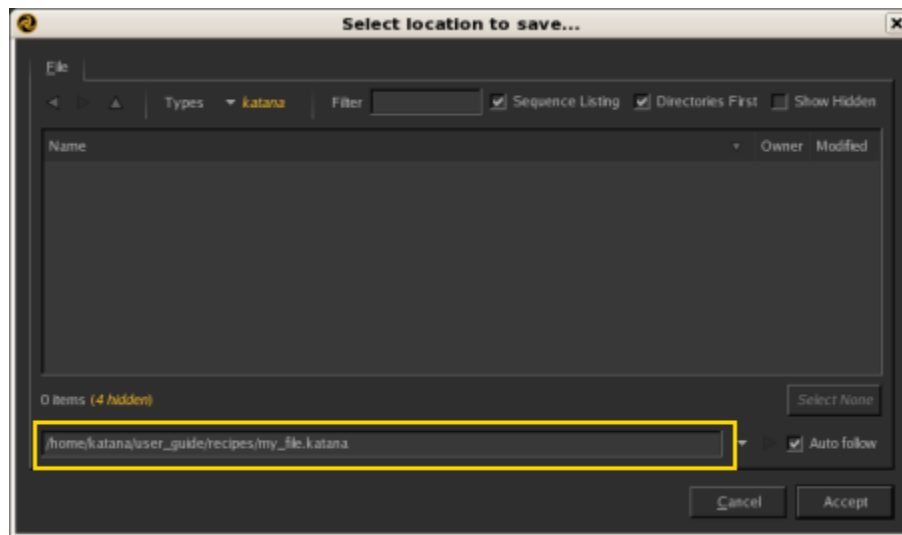
Saving to a New File

To save your current Katana project to a new file:

1. Select **File > Save As...** (or press **Ctrl+Shift+S**).

The **file browser** displays.

2. Navigate to the directory to save the file.
3. Add the filename to the text field below the main window.



4. Click **Accept**.



NOTE: If you're using a custom asset management system, the dialog you see may be different.

Loading a Katana Project

To load a Katana project:

1. Select **File > Open...** (or press **Ctrl+O**).
2. If needed, click **Load New Project** in the **Unsaved Changes** dialog window to confirm.
3. Select a Katana project from the **file browser** (see [Using the File Browser](#) below).
4. Click **Accept**.

Loading a Recently Saved Katana Project

To load a recent Katana project:

1. Select **File > Open Recent > ...** and select from one of the previously saved projects in the list.

2. If you have a project open already and you've made unsaved changes you don't wish to keep, click **Don't Save**, in the **Open** dialog to continue.



TIP: You can clear the list of recently opened projects by selecting **File > Open Recent > Clear Menu**.

Reverting Back to the Last Save

You can revert back to the last time you saved, to do so:

1. Select **File > Revert**.
2. Click **Revert Scene** in the **Unsaved Changes** dialog window to confirm.

The Katana project reverts back to the last save.

Importing a Katana Project

To import a Katana project into the current project:

1. Select **File > Import...** (or press **Ctrl+I**).
2. Select a Katana project from the **file browser** (see [Using the File Browser](#) below).
3. Click **Accept**.

The imported project's nodes float with the cursor inside the **Node Graph**.

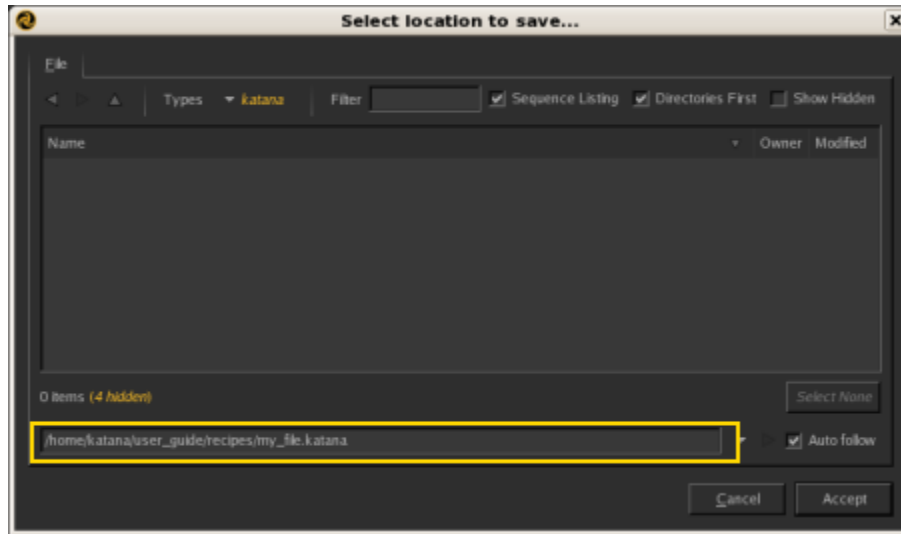
4. Click somewhere within the **Node Graph** to place the imported project at that location.

You can also import a Katana project as a LiveGroup. For more information on LiveGroups, and for help on how to import a project as a LiveGroup, see the [LiveGroups](#) chapter.

Exporting a Katana Project

Exporting from Katana gives you the ability to do the equivalent of **File > Save As...** but for a limited number of nodes. To export part of the current project:

1. Select the nodes you wish to export.
2. Select **File > Export Selection...** (or press **Ctrl+E**).
The **file browser** displays.
3. Navigate to the directory to export the file.
4. Add the filename to the text field below the main window.



5. Click **Accept**.

Changing a Project's Settings

Projects' settings are shared between each of the recipes created within that project. These can all be changed from within the **Project Settings** tab.

Setting	Description
inTime	The starting frame number for the timeline.
outTime	The ending frame number for the timeline.
currentTime	The current frame number.
timeIncrement	Changes the frame increment for the move forward and backwards icons in the timeline.
katanaSceneName	
resolution	The default resolution for 2D source files, such as ImageColor. Not used for the rendering of 3D scenes, they use the RenderSettings node instead.
plugins	
asset	The asset manager to use (defaults to File).
fileSequence	Plug-in to determine how to interpret a file sequence.

Setting	Description
compDefaults > fileIn	
missingFrames	How an ImageRead node behaves when a frame is missing.
inMode	What an ImageRead node displays for frames before its first frame.
outMode	What an ImageRead node displays for frames after its last frame.
compDefaults	
useOverscan	Whether to use overscan when rendering. Overscan is extra pixel information around the main render window.
viewerSettings	
normalsDisplayScale	Changes the size of normals when displayed in the Viewer tab.
proxyCacheSize	Number of proxy geometry objects to keep in memory.
monitorSettings	
overlayColor	Color to use when displaying alpha overlays.
ROI	
left	Left coordinate of the region of interest rectangle.
bottom	Bottom coordinate of the region of interest rectangle.
width	Width of the region of interest rectangle in pixels.
height	Height of the region of interest rectangle in pixels.

Dealing With Assets

Katana has been designed from the ground up to work within an asset based production environment. In fact, the philosophy behind Katana - the non-destructive recipe based approach - works to its fullest when used with assets that change and update in an iterative workflow. The decoupling of asset creation and their use in shots, allows a team to work in parallel.

Whether in a small, medium, or large studio, an asset management system helps maintain the large number of assets and revisions that artists create and use.

With its extensible **Asset Management API**, Katana can be made to slot into any production workflow that incorporates an asset management system. Examples of how to incorporate an asset manager using the Asset Management API are included with the Katana install. A full explanation of this process goes beyond the scope of this guide. For all examples within this guide, we assume you are using the **File** asset manager that ships as the default with Katana. For further details on the asset manager employed by your facility, consult your pipeline manager.

Selecting an Asset Manager

By default, Katana uses the file system to store assets. But Katana has the ability to plug into a studio's asset management system through its Asset Management API. Connecting Katana using this system is beyond the scope of the *Katana User Guide* and you should consult your pipeline manager and the technical guide that accompanies the installation for further information (the *Katana Technical Guide* is found under **Help > Documentation**).

Once connected, you can change the asset manager from within the **Project Settings** tab.

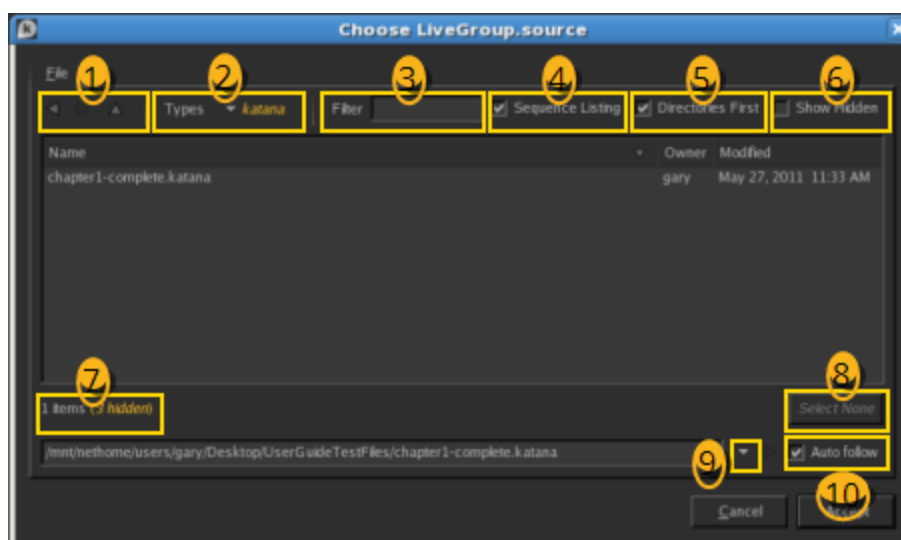
Changing the Current Asset Manager

You can select which asset manager to use by doing the following:

1. In the **Project Settings** tab, click the **plugins > asset** dropdown.
2. Select the asset manager from the filterable list.

Using the File Browser

The file browser is the basis for the **File** asset manager.

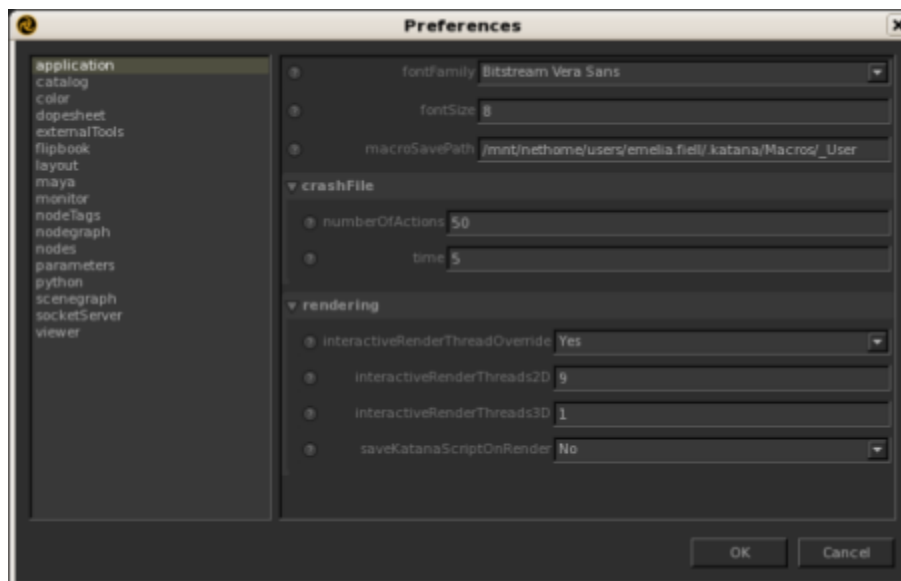


The file dialog

1. To step through the file browser's history, click the left and right arrows. To move up a directory, click the up arrow.
2. To change the filter controlling which files are displayed within the file dialog, click the dropdown next to **Types**.
3. Type in the **Filter** field to narrow the list of items within the main area. Only items that contain the string you entered are displayed.
4. Enable **Sequence Listing** to display image sequences as a single item. If disabled, image sequences are displayed as individual files.
5. Click the **Directories First** checkbox to toggle whether directories are displayed at the top or mixed in with the other files.
6. Click the **Show Hidden** checkbox to toggle whether hidden files are displayed.
7. The count for displayed (and filtered) items is displayed on the left below the main window.
8. To deselect all items, click **Select None**.
9. To quickly navigate to recent and parent directories, click the down arrow next to the filename.
10. To have Katana automatically update the main window as you navigate, enable **Auto follow**. If **Auto follow** is off, use  to navigate into a directory.

Autosaves

Your autosave preferences can be set under **Edit > Preferences > application > crashFile**.



The **crashFile** gives you two options, **numberOfActions** and **time**, to choose how frequently you want an autosave of your current scene to take place.

1. **numberOfActions** - specifies the number of actions before automatically saving the current project to a file from which the project can be restored after a crash.

2. **time** - specifies the time in minutes before automatically saving the current project to a file from which the project can be restored after a crash.



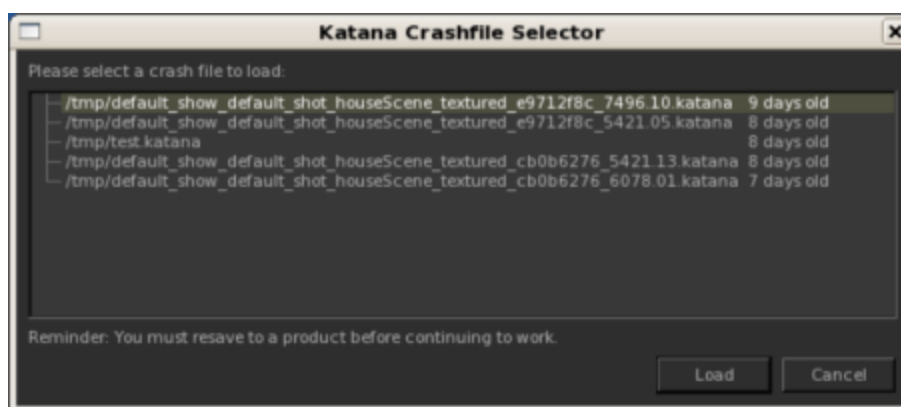
WARNING: Setting either preference to zero disables the corresponding autosave trigger. If both are set to zero, no autosave files are created.



NOTE: After an autosave file has been created another one cannot be saved until 15 seconds have past, even if the conditions set by **numberOfActions** or **time** are met.

Loading an Autosave File

To load an autosave file, launch Katana using the **--crash** command-line option, this opens the **Katana Crashfile Selector**.



From here you can select the autosave you would like to **Load**, note the most recent autosave is at the bottom of the list. If the crashed scenefile had not yet been saved, Katana calls the resulting autosave 'Untitled'.

Working with Nodes

Nodes are the basic building blocks of a Katana recipe. You create and connect nodes to form a tree of the operations you want to perform.

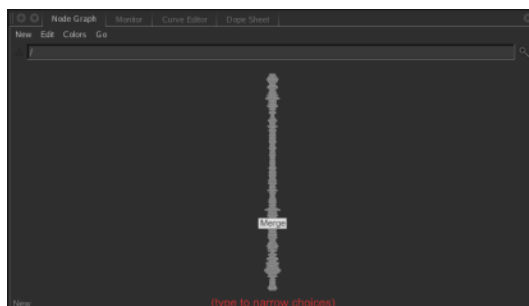
Adding Nodes

Add nodes to the **Node Graph** using the **Tab** menu, the **New** menu, or the right-click menu.

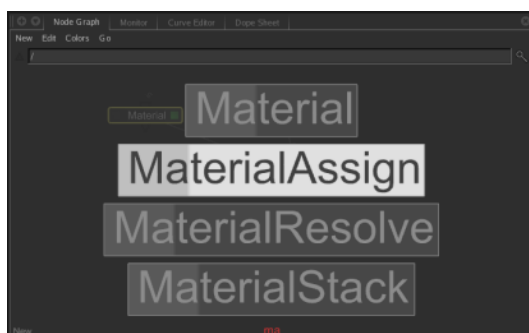
Adding a Node Using the Tab Key Menu

1. With the mouse over the **Node Graph** tab, press the **Tab** button.

Katana displays a list of all available nodes.



2. Narrow the list of nodes by either:
 - typing the starting letters of the node name, or
 - typing the capital letters that make up the node name (for instance, typing **MA** for the MaterialAssign node).



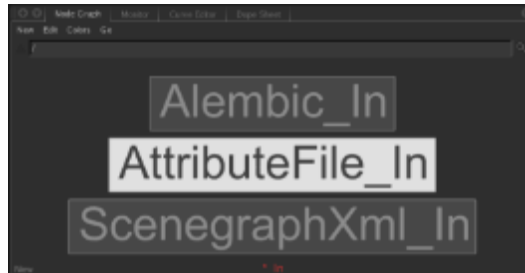
3. To select the node you want to add from the list, either:
 - click it, or

- scroll to it with the **Up** and **Down** arrow keys and press **Return**.

4. Click on an empty space in the **Node Graph** to place the node.



TIP: To add another copy of the last node created using this method, simply press **Tab** and then **Return**. Katana accepts wildcards while typing the name of the node to create. For instance, ***_In**.



Adding a Node Using the New Menu

1. In the **Node Graph** tab, click **New** and select the node you want to add.
2. Click on an empty space in the **Node Graph** to place the node.

Adding a Node Using the Right-Click Menu

1. Right-click on the **Node Graph** (or press **N**) and select the node you want to add from the menu.
2. Click on an empty space in the **Node Graph** to place the node.



TIP: While the node is floating with the mouse cursor, you can cancel the nodes creation by pressing **Esc**. To have Katana automatically connect the new node to the currently selected node, check the option **Edit > Auto Connect New Nodes Based On Selection** within the **Node Graph**.

Instead of placing the node and then connecting it, you can connect the node straight into the node tree by either:

- clicking on a connection, or
- clicking on another node's input or output, followed by clicking an empty space in the **Node Graph**.

Selecting Nodes

Katana offers a number of options for selecting nodes. Selected nodes are highlighted in yellow.

Selecting a Single Node

Click once on the node.

Selecting Multiple Nodes

Press **Shift** while clicking on each node you want to select.

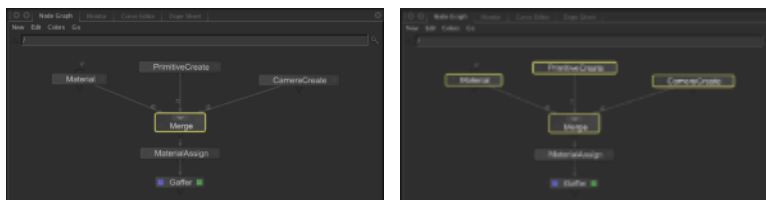
OR

Drag on the **Node Graph** to draw a marquee. Katana selects all nodes inscribed by the marquee.

Selecting all Nodes Upstream

You can select all the nodes upstream of the currently selected node(s). To do this:

1. Click on a node.
2. Press **Ctrl+Up Arrow**.

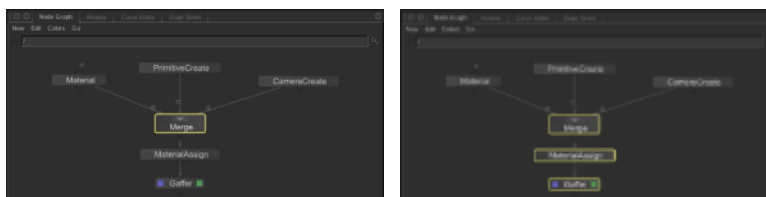


Katana selects all nodes that feed data to the selected node.

Selecting all Nodes Downstream

You can select all the nodes downstream of the currently selected node(s). To do this:

1. Click on a node.
2. Press **Ctrl+Down Arrow**.



Katana selects all nodes downstream from the selected node.

Adding to a Selection

Shift+click to select more nodes without clearing the current selection.

Deselecting a Node

To deselect a node:

Shift+click.

Connecting Nodes

As you build up a scene, you'll need to create connections between nodes or change the connections that already exist. Any nodes that are not connected to the overall node tree do not have any effect.

Nodes have input and output ports that are used to connect one node to another. Input ports are rectangles, usually located at the top of a node. Output ports are triangles, usually located at the bottom.

Connecting a Node into the Recipe

There are a number of different ways to connect a node into the recipe, you can:

1. Click the output port of the first node you want to connect.
2. Drag the resulting arrow to the input port of the second node.
3. When the input highlights in yellow, release the mouse button.

OR

1. Hover the cursor over the first node you want to connect.
2. Press the **Backtick** key (`) once.
3. Hover the cursor over the second node and press the **Backtick** key again.

OR

1. Drag one node over the input or output of a second node, and release the mouse button to establish a connection.
2. Click on an empty space in the **Node Graph** to then place the node there.

Adding a Node Between Two Other Nodes

1. Drag the node into the space between two already connected nodes.

When the cursor is over the connection, the connection becomes active (turns yellow).

2. Release the node you are dragging.

It automatically wires itself into the network between the two nodes.

Removing a Node from the Recipe

There are two different ways to disconnect a node without deleting it:

- remove its input/outputs manually, or
- extract it, which removes all connections and attempts to repair the recipe by connecting the nodes that are around the extracted node.

Disconnecting a Node

To disconnect a node, drag the head or tail of the connecting arrow to an empty area of the workspace.

Extracting a Node

You can remove all the connections to a node, extracting it from any recipes without deleting it. To extract a node:

1. Select the node you wish to extract.
2. In the **Node Graph**, select **Edit > Extract Selected Nodes** (or press **X**)

This removes all connections from the selected node, extracting it from the recipe.

Tidying the Recipe with a Dot Node

Dot nodes are used to help tidy a recipe and make the flow of the connections clearer.

They also have a unique ability in that disabling a Dot node ignores the contribution of all the nodes upstream.

To insert a Dot node:

1. Decide where to place the Dot node by:
 - selecting the node before the connector you want to bend, or
 - hovering the mouse over the connection you wish to bend.
2. Press **.** (full stop) to create a Dot node.
3. Drag the Dot node as necessary to reposition the connections.



TIP: You can also create the Dot node in the same way as any other node (through the **Tab** menu, **New** menu, or right-click menu) and connect it manually.



TIP: You can create a Dot node while connecting two nodes. With the connector connected at one end, press . (full stop) to place the Dot at the current mouse location, then continue as normal.

Replacing Nodes

To replace one node with another you can use the **R** key to replace a node using the same **Tab** key menu.

To replace a node using the **R** key:

1. In the **Node Graph**, select the node you want to replace.
2. Press **R** and start typing the name of the node you want to create.
Katana displays a list of matches.
3. To select the node you want to add from the list, either:
 - click on it, or
 - scroll to it with the **Up** and **Down** arrow keys and press **Return**.

The new node replaces the selected node in the **Node Graph**.

Copying and Pasting Nodes

To copy, paste, and perform other editing functions in the node tree, you can use the standard editing keys (for example, **Ctrl+C** to copy, and **Ctrl+V** to paste). Copied nodes inherit the values of their original, but these values, unlike those in cloned nodes (see below), are not actively linked - that is, you can assign different values to the original and the copy.

Copying Nodes to the Clipboard

1. Select the node or nodes you want to copy.
2. In the **Node Graph**, select **Edit > Copy** (or press **Ctrl+C**).

Pasting Nodes from the Clipboard

In the **Node Graph**, select **Edit > Paste** (or press **Ctrl+V**).

Katana adds the nodes to the scene.

Cutting Nodes from the Node Graph

1. Select the node or nodes you want to cut.
2. In the **Node Graph**, select **Edit > Cut** (or press **Ctrl+X**).

Katana removes the node(s) from the scene and writes the node(s) to the clipboard.

Cloning Nodes

You can clone nodes and place them elsewhere in a recipe. Cloned nodes inherit the values of their parent, but unlike copied nodes, they also maintain an active link with their parents' values. If you alter the values of the parent node, the clone automatically inherits these changes.

To clone nodes:

1. Select the node or nodes you want to clone.
2. In the **Node Graph**, select **Edit > Clone**.

Katana clones the node or nodes and creates an expression between each parameter of the parent node and that of the clone. Any change on the parent is therefore reflected in the child.

To declone nodes:

1. Select the node or nodes you want to declone.
2. In the **Parameters** tab, select  > **Reset Parameters**.

Katana removes the clone status of the selected nodes and resets all its parameters to the nodes' defaults.

Disabling Nodes

Disabling and Re-Enabling Nodes

You can toggle a node between enabled and disabled. To toggle whether a node is enabled:

Hover over the node and press **D**.

OR

1. Select the node or nodes.
2. In the **Node Graph**, select **Edit > Toggle Disabled State of Selected Nodes** (or press **Alt+D**).



Deleting Nodes

Deleting Selected Nodes

1. Select the node or nodes you want to delete.
2. Press **Delete**.

Katana removes the node(s) from the scene.

Deleting All Nodes Not Contributing to the Current Scene Graph

In the **Node Graph**, select **Edit > Delete All Non-Contributing Nodes**. Disabled nodes that would contribute if enabled are not deleted.

Navigating Inside the Node Graph

As recipes grow in complexity, you need to be able to move between clusters of nodes quickly. Katana offers various methods for doing so.

Panning

Middle-click and drag the mouse pointer over the workspace. The recipe moves with your pointer.

Zooming

You can zoom in on or out of the recipe in a number of ways.

Zooming In

Move your mouse pointer over the area you want to zoom in on, and press **+** (**Plus** key) repeatedly until the workspace displays the recipe at the desired scale.

OR

Alt+left/right-click and drag right.

OR

Move the mouse pointer over the area you want to zoom in on, and scroll up with the mouse wheel.

Zooming Out

Move your mouse pointer over the area you want to zoom out from, and press - (**Minus** key) repeatedly until the workspace displays the recipe at the desired scale.

OR

Alt+left/right-click and drag left.

OR

Move the mouse pointer over the area you want to zoom out from, and scroll down with the mouse wheel.



NOTE: In many Linux windows managers, the **Alt** key is used by default as a mouse modifier key. This can cause problems in 3D applications where **Alt** is used for camera navigation in 3D environments.

You can use key mapping to assign the mouse modifier to another key, such as the  (**Super** or **Meta**) key, but the method changes depending on which flavor of Linux you're using. Please refer to the documentation on key mapping for your particular Linux distribution for more information.

Fitting Selected Nodes in the Node Graph

To fit selected nodes in the **Node Graph**, press **F**. If no nodes are selected then the entire node tree fills the **Node Graph**.

Fitting the Node Tree in the Node Graph

To fit the entire node tree in the **Node Graph**, press **A**.

Organizing Nodes in Group Nodes

Group nodes are used to group together a number of nodes into a single node, which can help to simplify the Node Graph.

Creating Group Nodes

To create a Group node, in the **Node Graph** tab, select a number of nodes and press **Ctrl+G**.

A new Group node, with the previously selected nodes as its children, is created. Any connections between selected

nodes are preserved.

To create an empty Group node:

1. In the **Node Graph** tab, either:
 - Press **Tab** and select **Group** from the node list,
 - Right-click and select **Other > Group** from the menu, or
 - From the **Node Graph** tab's menu, navigate to **New > Other > Group**.

The Group node floats with the cursor.

2. Click inside the **Node Graph** tab to place it at that location.
A new empty Group node is created.

You can duplicate Group nodes like any other node, which also creates duplicates of any child nodes.

Navigating in Hierarchies of Group Nodes

Group nodes are similar to folders in a file system; they can be used to group other nodes, including other Group nodes, thereby creating hierarchies of nested Groups.

A breadcrumbs bar at the top of the **Node Graph** tab uses Node Buttons to represent the Group node whose contents are shown in the tab, as well as all of its ancestor Group nodes. You can click any Node Button in the breadcrumbs bar to enter the corresponding ancestor Group node.

To enter a Group node, either:

- Select the Group node to enter and press **Ctrl+Return**,
- **Ctrl**+middle-click the Group node to enter,
- Select the Group node to enter, and navigate to **Go > Enter Selected Group** in the **Node Graph** tab's menu.
- Click the Node Button that represents the node in the breadcrumbs bar, or
- Drag the Group node into the breadcrumbs bar.

To leave a Group node that was entered, either:

- Press **Ctrl+Backspace**,
- Click the  icon in the title bar of an entered Group,
- Click the Node Button that represents the parent node of the Group node (if any),
- From the **Node Graph** tab's menu, navigate to **Go > Up**,
- Click the Node Button that represents the parent node of the Group node (if any) in the breadcrumbs bar, or
- Click the **Go To Root** button in the breadcrumbs bar (if the Group node exists in the root level of the node graph).

To jump up to the root level of the node graph, either:

- Press **Ctrl+Shift+Backspace**,
- From the **Node Graph** tab's menu, navigate to **Go > To Root**, or

- Click the **Go To Root** button in the breadcrumbs bar.

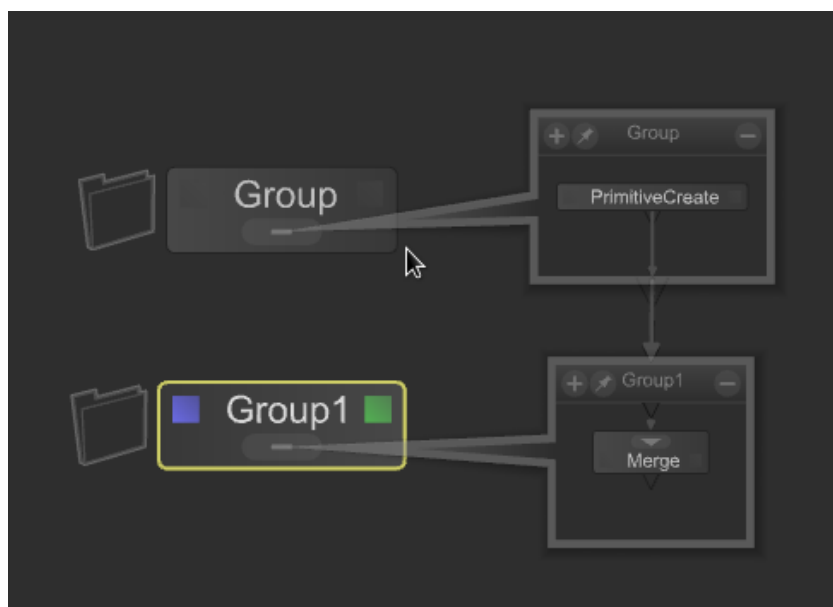
Connecting Group Nodes

Group nodes can have any number of input and output ports, to which nodes from outside of the Group can be connected, in order to connect to nodes inside the Group.

The simplest way to create input or output ports is to expand the Group node bubble to show its contents. Nodes from outside of the Group can then be directly connected to nodes inside the Group, which automatically creates the input or output port on the Group as required.



NOTE: For more on Group nodes, and their incoming and outgoing connections, see the *Group Nodes* section in the *Katana Technical Guide*.



Editing Group Nodes

Group nodes do not provide any parameters by default. However, you can add custom parameters to Group nodes, and link parameters of child nodes inside the Group nodes to those parameters through parameter expressions. That way, a Group node's parameters can act as the interface of the Group as a whole.

In order to edit the parameters of a Group node, either:

- Click the green edit flag on the Group node in the **Node Graph** tab,
- Move the pointer over the Group node, and press the **E** key, or
- Select the Group node, and press **Alt+E**.



NOTE: See the section on [Adding User Parameters](#) for more information.

Node Buttons

Node Buttons are used in the UI to represent nodes that exist in the current project's node graph. For example, a Node Button in the **Scene Graph** tab represents the currently viewed node, and Node Buttons in the **Parameters** tab represent the nodes whose parameters are edited.

Node Buttons show the name of the node they represent, show a view and/or edit icon if the node has its view and/or edit flags set, and use the color of the node for the button's background, if a node color is set.

Node Button Keyboard Shortcuts

Node Buttons have been designed to mimic the ways you can interact with nodes in the **Node Graph** tab. With the pointer over a Node Button, you can use the following keyboard shortcuts to make changes to the node represented by the Node Button:

Keyboard Shortcuts	Action
E	Sets the edit flag on the node. This shows the parameters of the node in the Parameters tab.
V	Sets the view flag on the node. This shows the scene graph that is produced by the node in the Scene Graph tab.
Shift + E	Toggles the edit flag of the node.
Shift + V	Toggles the view flag of the node.
D	Toggles the disabled state of the node.
Return, Enter, or F2	Renames the node.
Ctrl + Left-click	Reveals and selects the node in the Node Graph tab.
Middle-click + drag	Drags a representation of the node.
Right-click	Opens the same context menu that can be opened for a node in the Node Graph tab.

Improving Readability and Navigation with Backdrop Nodes

You can use Backdrop Nodes to help document your recipes, making them easier to read and navigate. They can be placed at the side of important nodes to explain their use for future users, around a collection of nodes that perform a particular function, or just as a title for your entire recipe. How you use them is up to you!

Creating a Backdrop Node

A Backdrop Node is created in the same way as any other node, through the **Tab** key menu, the right-click menu, or with the **New** menu within the **Node Graph**. As well as these methods you can also create a Backdrop Node around a number of nodes using the method below.

To fit a Backdrop Node around the currently selected nodes:

1. Select the nodes the Backdrop Node is to encompass.

A minimum of two nodes must be selected.

2. Select **Edit > Fit Backdrop Node to Selected Nodes**.

If you select a Backdrop Node with the selected nodes, Katana uses that Backdrop Node, otherwise a new Backdrop Node is created.

Editing a Backdrop Node

To change the parameters of a Backdrop Node:

1. Double-click within the horizontal lines at the top of the node.

This brings up the **Edit Backdrop Node** dialog.



2. In the dialog you can:
 - Enter or edit the text in the main text box.
 - Change the size of the text with **fontScale**.
 - Change the background color.
 - Toggle whether this Backdrop Node should be part of the jump-to menu with **Show In Bookmarks** (See [Navigating with Backdrop Nodes](#)).
 - Toggle whether this Backdrop Node should be drawn behind other nodes with **Send to Back**.
3. Click **Ok** to save changes.

Resizing a Backdrop Node

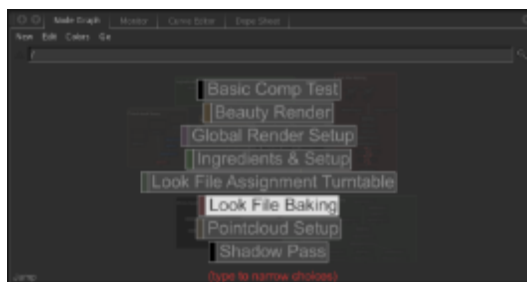
You can resize a Backdrop Node by dragging from the bottom right corner.

Navigating with Backdrop Nodes

One extremely useful function of Backdrop Nodes is their ability to act as jump to points throughout a project.

1. In the **Node Graph**, select **Go > Jump to Bookmark** (or press **J**) to bring up the Backdrop Nodes jump to menu.

Katana displays all the Backdrop Nodes that have the bookmark flag enabled with their background color displayed to the left.



TIP: The first line of a Backdrop Node is used as its title for the **Jump to Bookmark** menu.

2. Start typing the name of the node you wish to navigate to.
This narrows down the displayed list.
3. To select the Backdrop Node to navigate to, either:
 - click on it, or
 - scroll to it with the **Up** and **Down** arrow keys and press **Return**.

Selecting Nodes Within a Backdrop Node

You can select all nodes within the bounds of a Backdrop Node (as well as the node itself) by **Ctrl**+clicking within the two horizontal bars at the top of the node.

Locking and Unlocking Backdrop Nodes

To lock Backdrop Nodes so they can't be edited or selected, select **Edit > Lock All Backdrop Nodes**. All Backdrop Nodes are locked, but if you create a new Backdrop Node it is not locked.

To unlock all Backdrop Nodes, select **Edit > Unlock All Backdrop Nodes**.

Editing a Node's Parameters

Each node has parameters that alter how the node behaves within the recipe. These parameters can be changed within the **Parameters** tab.

A parameter's value comes from one of three things:

- A constant.
For example: 9, test, or **/root/world/cam/camera**
- An expression.
For example: 16-3, scenegraphLocationFromNode(getNode('CameraCreate')), or getNode('CameraCreate').fov. See [Appendix B: Expressions](#).
- A curve: only available for numeric inputs. See [Animating Within Katana](#).

Accessing a Node's Parameters

To edit a node's parameters, they need to be in the **Parameters** tab. To do this, you can:

1. Select the node(s) whose parameters you want to edit.
2. In the **Node Graph**, select **Edit > Edit Selected Nodes** (or press **Alt+E**).

OR

1. Hover the mouse pointer over the node you wish to edit.
2. Press the **E** key.

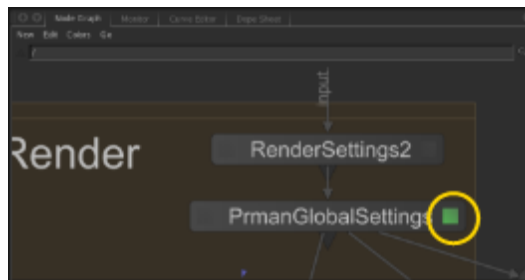
OR

Click within the faint square to the right of a node.

OR

Double-click on a node. This also sets the current scene graph view to that node. See [Using the Scene Graph](#).

A node that has its parameters in the **Parameters** tab has a green square on the right hand side.



Opening and Closing a Node's Parameters

Once a node's parameters are visible within the **Parameter** tab they are grouped with the node type and name at the top. This can be opened and closed with the  /  icons next to the node type.

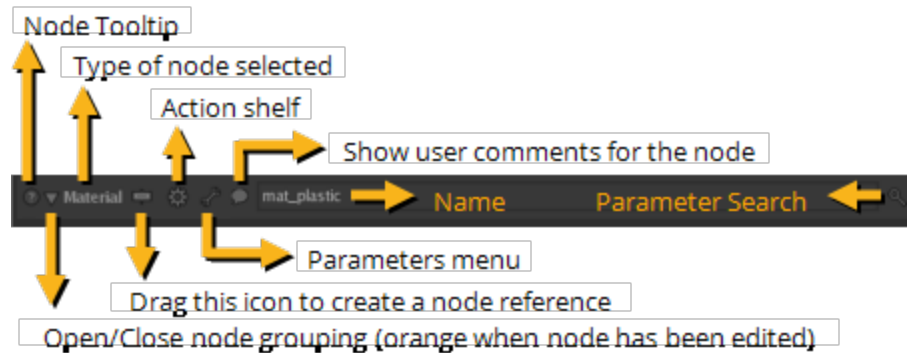


NOTE: If the **Parameter** tab is not visible you can either:

Add it to a pane by clicking the  icon on the relevant pane and selecting **Parameters**,
or
create a new floating pane, by clicking **Tags > Parameters**.

Default Parameters Tab Icons

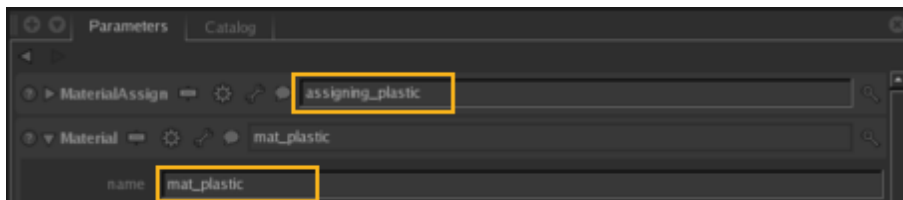
Nodes displayed in the **Parameters** tab have a number of default icons.



Changing a Node's Name

Different nodes have their name edited in different ways. The two most common places to get the node name are:

- The name in the input field at the top, next to the node type.
- One of the parameters, for instance the **passName** in the **Render** node, or the **name** in the **Material** node.



Changing a Node's Parameters

Each parameter type has a control associated with it. How to make changes to some of the more common parameter types is listed below.

Changing a Numeric Value

You can change a numeric value by:

- Entering a new value in the input field.
- Pressing the **Up Arrow** key to increment its value, or **Down Arrow** to decrement.
- Click and drag on the parameter name, also known as scrubbing. Dragging to the left decreases the value, and dragging to the right increases.



TIP: To make the changes coarser, hold down the **Shift** key while scrubbing, to make them finer, hold down the **Ctrl** key. Pressing **Shift** with the up and down arrows makes the change coarser, or pressing **Ctrl** makes it finer. Also, to change the increment and decrement amount, right-click and select the sensitivity from the **Sensitivity** menu.

Changing a Color Value

Use the color picker or the pixel probe to change the color.

Changing the Value of a Dropdown Menu

To change the value in a dropdown menu:

1. Click on the dropdown menu.
2. Then, either:
 - Click on the new value from the list.
 - Use the **Up** and **Down Arrow** keys to highlight the new value and press the **Return** key.

Changing a Text String

A string can be used to represent a texture name, scene graph location, node name, or whatever a plug-in may need. Depending on what it is representing it can be displayed in a number of ways. These can be:

- a plain text input field, or
- a scene graph location, or
- a filename.

Manipulating a Scene Graph Location Parameter

Scene graph location parameters are used to either point to where a new location is inserted into the scene graph or to reference an existing location.

When the node creates a new location within the **Scene Graph** tab, the  icon presents you with common path prefixes to aid in placing the new location. When the node modifies an existing location, the  icon allows you to get the path from either:

- the current **Scene Graph** tab selection, or
- the current **Node Graph** node selection.

If you choose the second option, Katana creates an expression that points to the scene graph location created by the selected node.

To find the location that the parameter references and select it within the **Scene Graph** tab, click  and select **Select In Scenegraph**.



NOTE: Some nodes that create scene graph locations can be linked to a parameter via an expression so whatever scene graph location is created by the node becomes the value of the parameter. To generate the link **Shift**+middle-click and drag from the node to the parameter.

Assigning Locations to a CEL Parameter

CEL parameters can be made up of one or more statements. Each statement can be one of three things:

- a path,
- a collection (a CEL statement stored on a scene graph location), or
- a custom CEL statement.

For more on valid CEL statements, see [Appendix C: CEL and Collections](#).

State Badges

Some parameters, and all attributes, have an icon to help you determine how the current value is being assigned. These are referred to as state badges.

Icon	What it means
	This parameter or attribute has not been set and is getting its value from a predefined default.
	This parameter is being forced to use the predefined default value.
	This parameter has a local change and is being set at this node.
	This parameter or attribute has been set and is not getting its value from the default. A parameter with this icon would have already been set further up the node tree.
	This attribute is inherited from a parent location further up the scene graph hierarchy.
	This parameter or attribute has an active reference to a parameter in another file. Changes to the other file update this parameter when reloaded.

Customizing Node Display

You can change how a node is displayed to improve clarity and readability and to provide additional information about a node's behavior.

Changing a Node's Background Color

To change a node's background color to one of the preset colors:

1. Select the node or nodes to change.
2. In the **Node Graph**, select **Colors**, and choose a color from the presets.



NOTE: If in the **Node Graph, Edit > Dim Nodes Unconnected to View Node** is selected, or a node is ignored, its background color does not change.

For information on changing node colors using Python, and changing the display color of all nodes of a particular type, see the *Custom Node Colors* chapter in the *Katana Technical Guide*.

To change a node's background color to a custom color:

1. Select the node or nodes to change.
2. In the Node Graph, select **Colors > Set Custom Color**, and choose a color from the **Color Picker** window.



NOTE: To reset a node's color back to the default, select the node, then choose **Colors > None**.

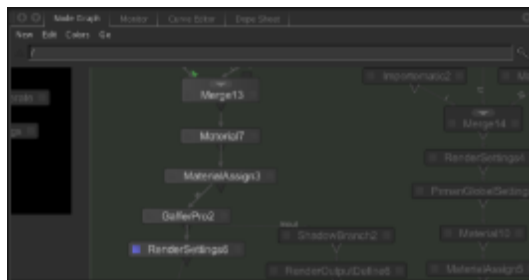
Dimming Nodes not Connected to the View Node

To improve visibility you can dim all nodes not relevant to the currently viewed scene graph. In the node graph, select **Edit > Dim Nodes Unconnected to View Node** or press **Alt+.** (full stop) in the **Node Graph** tab.

For instance with the Switch node, you can use the **Dim Nodes Not Contributing to Viewed Node** option in the **Edit** menu of the **Node Graph** tab to visualize which portion of the node graph has been selected by the Switch node.

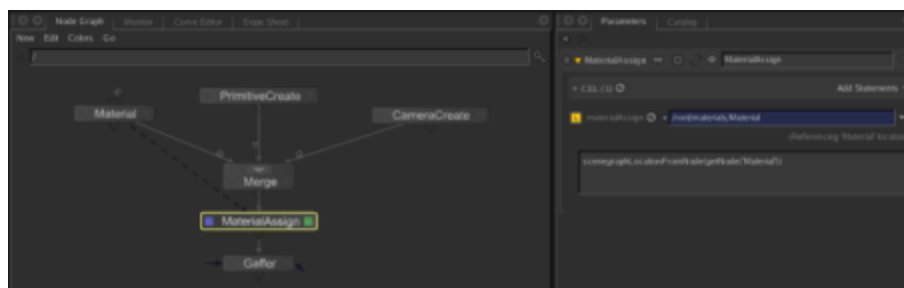


NOTE: The **Dim Nodes Not Contributing to Viewed Node** feature does not take into account nodes whose parameters are referenced in parameter expressions on nodes that are contributing to the currently viewed node.



Displaying Nodes Linked by an Expression

Some nodes are linked to other nodes through expressions. To display this relationship with a dark dashed line in the **Node Graph**, select **Edit > Show Expression Links** (or press **Q**) from within the **Node Graph** tab.



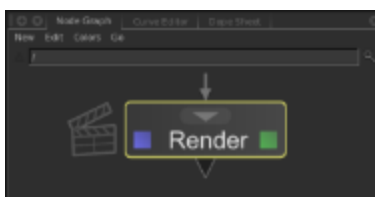
The **materialAssign** parameter of the MaterialAssign node uses an expression to get the scene graph location created by the Material node.

Drawing the Node Graph with Reduced Contrast

To reduce the contrast around nodes and their connections, in the **Node Graph**, select **Edit > Draw Graph with Low Contrast**. You can do this in conjunction with dimming unconnected nodes.

Aiding Project Readability with Node Icons

By default, some nodes have icons displayed to their left making it clearer what their function is. This behavior is toggled within the **Preferences** dialog.



To Toggle Node Icons:

1. Select **Edit > Preferences** to bring up the **Preferences** dialog (or press **Ctrl+, (comma)**).
2. Click **nodegraph** in the list on the left.
3. Change **showNodeIcons** to **Yes** to display the icons, or **No** to hide.
4. Click **Ok** to make the changes permanent.

Image Thumbnails

Thumbnails provide a guide to the image generated at a particular node within the recipe. Most 2D nodes can display thumbnails, as can the Render node. Although some nodes display thumbnails by default, others need it activated.

To Toggle Thumbnail Display for Thumbnail Capable Nodes:

Right-click and select **Display Thumbnail**.

To Update a Thumbnail:

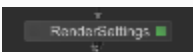
Right-click and select **Regenerate Thumbnail**.

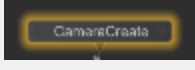


NOTE: Thumbnails don't update automatically!

Indicators on Nodes

There are several indicators that can display on the nodes in the **Node Graph**. The following table describes what each indicator means.

Indicator	What it means
	This node is selected.
	This node's parameters are being edited in the Parameters tab.

Indicator	What it means
	This node is being viewed. The scene graph generated up to this node is currently displayed in the Scene Graph tab.
	This node is disabled.
	Edits to the currently selected location using an interactive manipulator within the Viewer tab are fed back to this node.
	An error occurred in the processing of the scene graph at this node.
	An error occurred in the processing of the scene graph at a node within this node. Tip: To see which node, Ctrl +middle-click on the node to view inside.
	Edits to the currently selected location using an interactive manipulator within the Viewer tab are fed back to the node inside this node. Tip: To see which node, Ctrl +middle-click on the node to view inside.
	A node inside this node has its parameters being edited in the Parameters tab. Tip: To see which node, Ctrl +middle-click on the node to view inside.
	A node inside this node is being viewed. The scene graph generated up to that node is currently displayed in the Scene Graph tab. Tip: To see which node, Ctrl +middle-click on the node to view inside.

Using the Scene Graph

The scene graph is a hierarchical structure that represents the scene generated by stepping through the recipe up to the node in the **Node Graph** with the blue square. The node with the blue square is sometimes referred to as the **view node**, this is because the scene graph is just a view of the 3D scene generated up to that node.

The information within the **Scene Graph** tab contains (but is not limited to) geometry, materials, lights, cameras, and render settings. Each node within the **Node Graph** tab describes a step within the recipe, which adds, deletes, or modifies scene graph locations or scene graph data. Scene graph data is stored as attributes on locations.

Scene Graph Terminology

Name	Type				Lights
▼ /root	group	☐	☐	☒	
▼ /world	group	☐	☐	☐	
▼ /geo	group	☐	☐	☐	
└─ /primitive	polymesh	☐	☐	☐	
▼ /cam	group	☐	☐	☐	
└─ /camera	camera	☐	☐	☐	
▼ /materials	group	☐	☐	☐	
└─ /Material	material	☐	☐	☐	

The selected location has a path of **/root/materials**.

The location **/root** is the parent of **/root/materials**.

The location **/root/materials/Material** is a child of **/root/materials**.

The location **/root/world** is a sibling of **/root/materials**.

The location **/root/world/geo/primitive** is a leaf location. A leaf is a location with no children.

The locations **/root/world** and **/root/materials** are two branches from **/root**.

Locations within Katana have a special attribute called **type**. This attribute tells Katana what type of information to expect at that location. In the example above, there are five group locations and one geometry material location.

Viewing the Scene Graph

You can view the scene graph generated at any node within the **Node Graph**. This shows the 3D scene generated by the recipe up to that point. To view the scene graph at a particular node:

1. Select the node in the **Node Graph**.
2. In the **Node Graph**, select **Edit > View Selected Node**.

OR

1. Hover the mouse over the node.
2. Press the **V** key.

OR

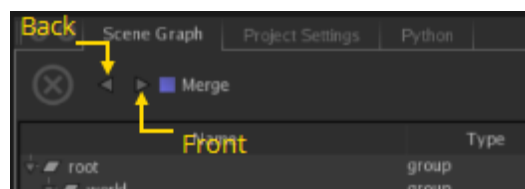
Click within the faint square to the left of the node.



NOTE: A blue square highlights the current node in the **Node Graph** tab, from which the scene graph is generated. This node is known as the **view node**. If the node moves off the screen, or is hidden within another node, its location is indicated by a small blue triangle.

Navigating the Scene Graph History

Katana keeps a history of the **view node** that can be traversed. To go back and forward through the history, use the icons in the upper-left area of the **Scene Graph** tab.



Manipulating the Scene Graph

Katana's **Scene Graph** tab is designed to work with scenes of almost unlimited complexity by only displaying the elements of the scene graph that are needed. By default the scene graph starts with its locations collapsed, so only **/root** is visible.

Selecting and Deselecting Locations in the Scene Graph

Selecting Multiple Scene Graph Locations

1. Select the first location.
2. **Shift**+click a second location.

Katana selects both locations and all in-between locations that are visible within the scene graph.

OR

1. Select the first location.
2. **Ctrl**+click the locations to add.

Selecting the Parent of the Selected Location(s)

1. Right-click on the selected location(s).
2. Select **Select > Select Parents**.

Selecting the Children of the Selected Location(s)

1. Right-click on the selected location(s).
2. Select **Select > Select Children**.

Selecting the Leaves of the Selected Location(s)

1. Right-click on the selected location(s).
2. Select **Select > Select Visible Leaves**.

Inverting the Selection with its Siblings

1. Right-click on the selected location(s).
2. Select **Select > Invert Selection**.

Selecting the Material Location Assigned to the Currently Selected Location

1. Select a location with a **materialAssign** attribute.
2. Right-click on the selected location.
3. Select **Select > Select Assigned Material Location**.

Katana selects the location of the material that is assigned at this location. That material location is stored in the **materialAssign** attribute.

Deselecting a Location

Use **Ctrl**+click.

Selecting Locations with the Search Facility

Katana scene graphs can get extremely complicated. To make it easy to find the location you need, Katana has a search facility.

To use the search facility:

1. Click  to bring up the search dialog.
2. To narrow the search results you can:
 - Select the type of locations to search for in the dropdown at the top of the dialog (**Selected**, **Pinned**, **Cameras**, **Lights**, and **All**), or
 - Type text in the **Filter** field to narrow the search to only include locations with matching text.
3. To select a location, select its path within the dialog.

OR

To select all the locations displayed in the dialog, click **Select All Matching**.



NOTE: Only locations that are exposed within the scene graph are searched.

Expanding the Scene Graph

Expanding the Scene Graph Completely Below a Location

1. Right-click on the location to expand.
2. Select **Expand All**.



WARNING: Use with caution on big scenes as it can be time consuming to expand the entire scene graph.

Expanding to a Limited Level of the Scene Graph

Assemblies, components, and lod-group (level of detail group) locations are special locations designed to help organize complicated scene graphs. They are explained in greater depth at [Making Use of Different Location Types and Proxies](#).

1. Right-click on the location to expand.
2. Select the level of the scene graph to expose:

- **Expand To > assembly, component or lod-group**

Expands the scene graph from the selected location until it reaches either an assembly, component, or lod-group location. If none are found down a scene graph branch, it expands to the leaf location.



NOTE: This is the same as **double-clicking** a scene graph location.

- **Expand To > component**

Expands the scene graph until it reaches a component location. If none are found down a scene graph branch, it expands to the leaf location.

- **Expand To > assembly**

Expands the scene graph until it reaches an assembly location. If none are found down a scene graph branch, it expands to the leaf location.

- **Expand To > lod-group**

Expands the scene graph until it reaches an lod-group location. If none are found down a scene graph branch, it expands to the leaf location.

Expanding a Location Only One Level

- Click  next to the location name.

OR

1. Right-click on the location to expand.
2. Select **Expansion > Open**.

Collapsing the Scene Graph

Collapsing a Location and All Its Children

1. Right-click on the location to collapse.
2. Select **Close All**.

Collapsing a Scene Graph Location

- Click  next to the location name.

OR

1. Right-click on the location to collapse.
2. Select **Expansion > Close**.

Collapsing the Scene Graph Completely

- Right-click on **/root** and select **Close All**.

OR

- Click  > **Clear Scene Graph State**.

This option doesn't just clear the scene graph, it also clears your selection and pins from the scene graph.

Bookmarking a Scene Graph State and Working Sets

Bookmarks can be used to save and restore Working Sets, and the currently-expanded and -selected parts of the scene graph, within a single session of Katana, and between sessions when Katana is closed and re-opened. This allows you to return quickly to a Working Sets location state or a scene graph state at any time, for example, if you want Katana to show a particular set of objects in the Viewer, or are interested in inspecting the attributes of a specific deeply-nested location.



TIP: Saving a Katana project also saves its bookmarks.

Bookmarks are especially useful when rendering only selected items in the **Scene Graph** tab (accessible from the **Render Only Selected Objects** menu bar button  or pressing **F7**). Saving the Working Sets, expanded state and current selections of the **Scene Graph** tab ensures that if you want to select a different location or click elsewhere in the scene graph, your current locations can be easily reloaded and aren't lost.

Options for creating, managing, importing, and exporting bookmarks are all contained in the **Scene Graph Bookmarks**  dropdown menu in the **Scene Graph** tab. Alternatively, you can also clear the scene graph state and Working Sets at any time by clicking  > **Clear Scene Graph State** or **Clear All Working Sets** in the dropdown menu.

Creating a Scene Graph Bookmark

1. Click  > **Create Bookmark....**

The **Create Scene Graph Bookmark** dialog displays.

2. Type a bookmark name in the **Bookmark name** field or select **Last File Save** from the dropdown menu.

The **Last File Save** option overwrites the **Last File Save** bookmark of the previously saved file with the expansion state, but not the selection state.

3. To create the bookmark within a folder, type the folder name in the **Create in folder** field.
4. Select the **Working Sets** to include within the bookmark:

- **liveRenderUpdates** - stores which locations are active when Live Rendering.
- **render** - stores which locations are rendered in Preview Rendering.

- **scenegraphExpansion** - stores which locations are open or closed.
- **scenegraphPinning** - stores which locations are pinned. For more on pins, see [Changing What is Shown in the Viewer](#).
- **scenegraphSelection** - stores which locations are selected.
- **viewerVisibility** - stores which objects are shown in the **Viewer** tab.

5. Click **Save** to create the bookmark.



NOTE: For more on Working Sets, see [Working Sets in Scene Graph Tab](#).

Deleting Unused Bookmarks

1. Click  > **Organize Bookmarks....**

The **Organize Scene Graph Bookmarks** dialog displays.

2. Right-click on the bookmark and select **Delete** to delete it.

3. To close the dialog again, click the **x** in the upper-right corner of the dialog.

The bookmark is successfully deleted and the dialog is closed.

Exporting the Project's Bookmarks

1. Click  > **Export Bookmarks....**

The **Export Scene Graph Bookmarks** dialog displays.

2. Choose a location and filename.

3. Click **Accept**.

A file containing all the exported bookmarks is saved.

Importing Previously Exported Bookmarks

1. Click  > **Import Bookmarks....**

The **Import Scene Graph Bookmarks** dialog displays.

2. Navigate to where the bookmarks file is saved to import.

3. Click **Accept**.

The bookmarks from the file are loaded into the project and can now be found in the bookmark  dropdown.

Viewing a Location's Attributes

To view the attributes stored at a location within the scene graph, select the location within the **Scene Graph** tab and the attributes display in the **Attributes** tab. The **Attributes** tab is read-only.

Changing What is Shown in the Viewer

The **Viewer** tab is a 3D representation of the scene currently open in the scene graph. Part of Katana's ability to deal with extremely complex scenes comes from it only loading scene graph data when it is needed. The **Viewer** tab only shows locations that are expanded in the **Scene Graph** tab, as well as any pinned locations that have been set.

Pinning can make navigating and organizing the Viewer easier by quickly showing/hiding specific locations. Pinned items are still visible in the Viewer, even when their locations are collapsed in the **Scene Graph** tab.



NOTE: If your **Viewer** is empty, it probably means that no locations with geometry have been expanded in the **Scene Graph** tab, and no pins have been set.

There are a couple options for pinning locations in the scene graph - you can pin selected locations or all visible leaf locations. When a location is pinned, it appears with a blue, active pin icon  next to it in the **Scene Graph** tab. If a location is pinned further down in the hierarchy, but the location is hidden in the scene graph because one of its ancestors is collapsed, then its nearest visible ancestor is shown with a gray, tilted pin icon . This denotes that there is a child pin somewhere in the hierarchy below that location. If you follow the "trail" of  icons through the scene graph, it leads you to the exact pinned location, shown with .

Pinning a Location or Locations

1. Select the location(s) you want to pin in the **Scene Graph** tab.
2. Right-click and select **Pin > Set Local Pin**.

The  pin icon appears next to the selected location(s).

Pinning All the Visible Leaves

First, ensure the locations you wish to pin are expanded in the **Scene Graph** tab.

1. Select the top-level location(s) for the leaves you want to pin.
2. Right-click and select **Pin > Pin Visible Leaves**.

Katana pins all the visible leaf locations below the selected location(s). The leaf locations are marked with  pin icons.

Clearing Selected Pins

1. Select the location(s) where you want to remove the pin.
2. Right-click and select **Pin > Remove Local Pin**.

The  pin icon is removed from the selected location(s).

Clearing Pins Below Selection

1. Select the top-level location(s) from where you want to remove lower-level pins.
The top-level locations show the  pin icon if they have pins on a lower-level location.
2. Right-click and select **Pin > Clear Pins Below**.
Katana removes any pins found on child locations beneath the selected location(s).

Rendering only Selected Objects

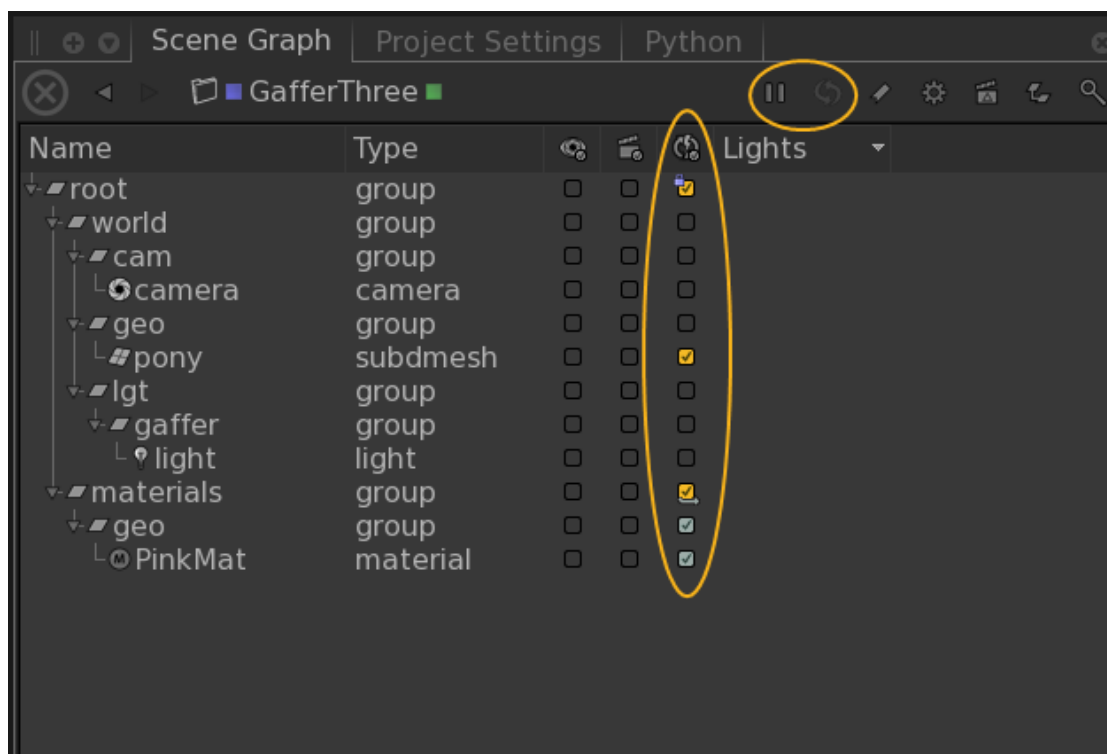
For speed it is sometimes preferable to only render a subset of the objects within a scene. To limit the objects being sent to the renderer, select the objects in the scene graph and click the Render only Selected Objects  icon.



NOTE: This is only for preview renders. Performing a disk render uses the entire scene graph.

Controlling Live Rendering in the Scene Graph

The **Scene Graph** tab allows you to select which lights or materials you wish to Live Render with the Live Render Updates  column and you can also choose how to trigger a Live Render with the **3D Update Mode**.



Using the Live Render Updates Column

In the **Scene Graph** tab, simply tick the relevant boxes in the Live Render Updates  column depending on which light and/or material changes you want to send to the renderer. For more information on the Live Render Updates  column, see [Global Options](#).

Using the 3D Update Mode

Select how to trigger Live Rendering updates with the following options:

3D Update Mode	Indicators	Actions
Manual mode		Changes to materials, lights, or geometry transformations don't trigger a Live Render update. To have the changes take effect, click the Trigger 3D Update button  .
Pen-up mode		Changes to materials, lights, or geometry transformations trigger a Live Render only when the mouse is released or a parameter change is applied.
Continuous mode		Changes to materials, lights, or geometry transformations, including some manipulations in the Viewer tab, trigger a Live Render.

For more information on the **3D Update Mode**, see [Global Options](#) in the [Rendering Your Scene](#) chapter.

Turning on Implicit Resolvers

Katana defers some procedures, such as a material copy, until they are needed by the renderer. This deferring has a number of positive results:

- It speeds up the initial scene graph generation.
- You can keep everything at a higher level making it easier to edit and override. For instance, you can change what material is at a given location rather than having to edit or override all the individual shader values.

Some examples of procedures that are deferred are:

- The copying of all the material details to a location.
- The copying of all the texture details to a location.

These deferred procedures are also known as implicit resolvers. To turn on implicit resolvers click .

Making Use of Different Location Types and Proxies

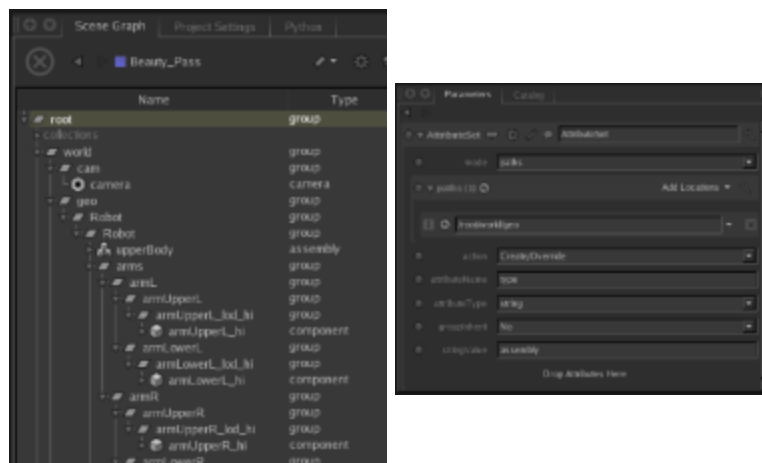
Only loading what is needed, when it is needed, is a key part of the philosophy of Katana. If you want to position the specular highlight on your main character, for instance, you don't need to load the entire scene. One way to avoid unnecessary loading is to define your scene with special hierarchies and proxies.

The hierarchical scene structure can be created using special location types. Special types that can be used are **assemblies**, **components**, **level-of-detail groups**, and **level-of-detail** locations.

Proxies enable you to get a good idea of a scene without opening up too much of the hierarchy. Placing proxies on assemblies and components enables you to open a hierarchy to convenient levels of scene complexity.

Using Assemblies and Components

Assemblies and components help you define convenient points of expansion for the scene graph. They are usually defined as part of the asset creation process, but you can also define them within Katana. An asset's hierarchy usually consists of an assembly and then below the assembly are other assemblies or components, and below the components is the full geometry data.



A scene graph example containing assembly and component locations.

A simple example of using an AttributeSet node to change the location type (which is simply the **type** attribute for a location) to **assembly**.

Working Sets

Working Sets provide a flexible way to work with particular locations and branches of a scene graph.

The main purpose of Working Sets is to decouple the expansion and selection states of scene graph locations in the **Scene Graph** tab from what's being drawn in the **Viewer** tab and from what's being rendered when rendering. Traditionally in Katana, the **Viewer** tab was closely linked to the **Scene Graph** tab. The Viewer displayed geometry of locations that were expanded in the **Scene Graph** tab, and you could choose a sub-set of objects to be rendered using the **Render Selected Objects Only** option. The expansion and selection states were therefore critical to an artist's workflow. The main purpose of Working Sets is to decouple the expansion and selection states of scene graph locations in the **Scene Graph** tab from what's being drawn in the **Viewer** tab and from what's being rendered when rendering.

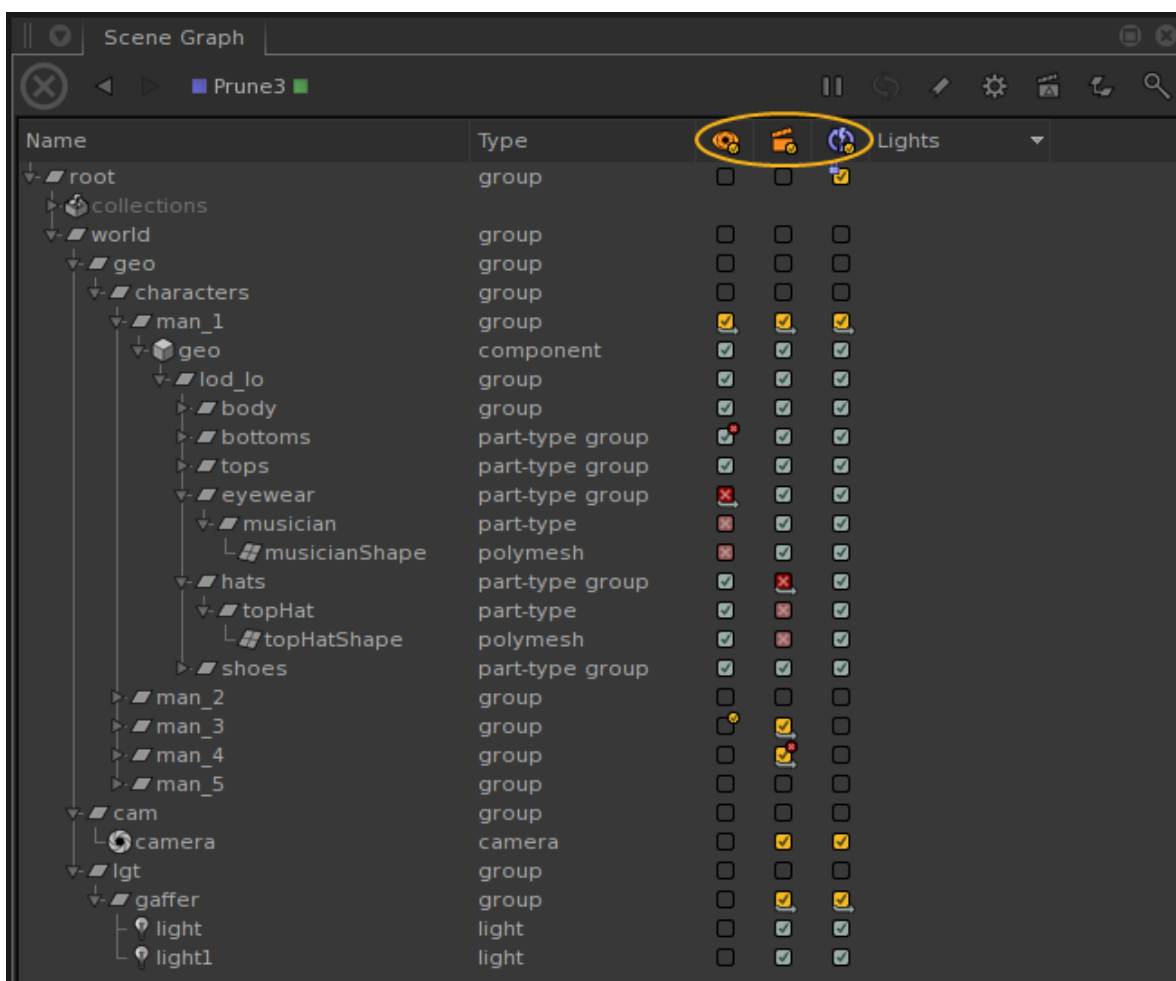
Working Sets are intended to allow artists to inspect a scene in the **Scene Graph** tab by expanding and collapsing branches at will, without incurring draw operations in the **Viewer** tab or render updates when Live Rendering. Artists are to be able to add and remove scene graph locations to and from specific Working Sets, and to use specific Working Sets in relevant UI operations.

Working Sets provide the ability to work with particular locations and branches of a scene graph with much more flexibility than is afforded by scene graph expansion, pinning, and selection. For example, a Working Set can be used to control which objects to render in Preview Renders and Live Renders. You can specify which scene graph locations are part of that Working Set in the **Render** column of the **Scene Graph** tab.

Working Sets in Scene Graph Tab

The **Scene Graph** tab contains columns for defining which locations to include or exclude in specific pre-defined Working Sets that are built into Katana:

Column	Description
 Viewer Visibility	Controls which objects to show in the Viewer tab.
 Render	Controls which objects to render in Preview Renders and Live Renders.
 Live Render Updates	Controls which objects trigger updates during a Live Render.



Screenshot of the **Scene Graph** tab showing various location states set for various locations in the pre-defined Working Set columns.

The states of scene graph locations in each Working Set are represented by icons in the corresponding Working Set column. The following table lists the icons that are used and the location states they represent:

Icon	Description
☐ - Empty	The location is neither explicitly included nor excluded in the Working Set, meaning it is not part of the Working Set.
☒ - Included	The location is explicitly included in the Working Set.
☒ - Included with Children	The location and all of its children are included in the Working Set, except those that are explicitly excluded.
☒ - Excluded	The location is explicitly excluded from the Working Set.

Icon	Description
 - Excluded with Children	The location and all of its children are excluded from the Working Set, except those that are explicitly included.
 - Included by inheritance	The location is included in the Working Set because one of its ancestors is Included with Children.
 - Excluded by inheritance	The location is excluded from the Working Set because one of its ancestors is Excluded with Children.

In addition to the above icons, the following icon decorations are used to indicate location states of children and/or restrictions on the states which a location is permitted to have:

Icon Decoration	Description
 - Children included	Children of the location are explicitly included in the Working Set.
 - Children excluded	Children of the location are explicitly excluded from the Working Set.
 - States restricted	Only certain states are allowed to be set for the location.

Interacting with Working Sets

Three pre-defined Working Sets can be manipulated through their columns in the **Scene Graph** tab: the Viewer Visibility Working Set, the Render Working Set, and the Live Render Updates Working Set.

Viewer Visibility Working Set

The **Viewer Visibility**  column allows you to interact with the Viewer Visibility Working Set that controls the visibility of objects in the **Viewer** tab. When the **Viewer Visibility** column is turned off, the Viewer displays locations depending on the scene graph expansion and pinned locations. When it is turned on, it displays locations included in the Working Set. You can turn on the Viewer Visibility Working Set by clicking on the **Viewer Visibility**  icon, either in the **Scene Graph** or **Viewer** tab. You can create arbitrary hierarchies of included and excluded scene graph branches. For example, you can include one location with all but one of its children.

You can also directly select an object in the **Viewer** tab and set a Working Set location state for it. Simply select one or more objects, then right-click and select a state from the menu.



NOTE: While the Viewer is following the Viewer Visibility Working Set (Viewer Visibility column is turned on), proxies and bounds are displayed only on leaf locations as defined by the Working Set, regardless of the existence of any child locations. Such leaf locations are directly set as **Included**, have no explicitly included children, and do not inherit inclusion. This allows proxy visibility to be determined without the need to cook child locations.

Render Working Set

The **Render**  column allows you to interact with the Render Working Set that controls which locations are rendered in Interactive Renders. You can turn on the Render Working Set by clicking on the Render  icon, either in the **Scene Graph** or **Monitor** tab.



NOTE: When the **Render Only Selected Objects** toggle is turned on, the locations that are currently selected in the **Scene Graph** tab are rendered, no matter what locations are part of the Render Working Set.

For more information, refer to [Rendering only Selected Objects](#).

Live Render Updates Working Set

The **Live Render Updates**  column allows you to interact with the Live Render Updates Working Set that controls for which locations updates are sent to the renderer when Live Rendering.

The **/root** location is always included, as Live Rendering requires updates to its attributes, notably `liveRenderSettings`, to be communicated to the renderer plug-in. The small blue lock  in the icon indicates that only some of the available location states can be set for the corresponding location.



NOTE: For more information on how to use the **Live Render Updates**  column, refer to [Using the Scene Graph](#).

Saving and Restoring Working Sets States

Bookmarks can be used to save and restore Working Sets, and the currently-expanded and -selected parts of the scene graph, within a single session of Katana, and between sessions when Katana is quit and re-started. This allows you to return quickly to a Working Set configuration or scene graph state at any time. For example, if you want Katana to show a particular set of objects in the Viewer, or are interested in inspecting the attributes of a specific deeply-nested location.



NOTE: To store the state of the Working Sets in your Katana project in a scene graph bookmark, refer to [Bookmarking a Scene Graph State and Working Sets](#)

Adding 3D Assets

Overview

The most common way to start a recipe is by defining the steps that bring in your 3D assets. Possible assets include static geometry, animated geometry in the form of a geometry cache, a particle cache, or an animated camera from a camera tracking package.

Katana's most common nodes for bringing in scene assets are:

- **PrimitiveCreate**

The PrimitiveCreate node contains a list of basic geometry shapes used in most 3D packages. These range from simple shapes such as planes and cylinders to teapots and gnomes.

- **CameraCreate**

A simple node designed to create a camera. You can also import cameras using the Alembic_In node.

- **Alembic_In**

The Alembic open standard has been adopted by Katana as its preferred means of asset interchange. Alembic is covered in more depth in [Adopting Alembic](#).

- **ScenegraphXml_In**

Katana can also bring in assets defined using XML. With Scene Graph XML you can define level of detail groups, assemblies, and components. For more on these and ScenegraphXml_In, see [Describing an Asset Using XML](#).

- **Importomatic**

The Importomatic is a one-stop-shop for bringing in assets. It has a plug-in structure enabling assets to be imported from different formats. It ships with plug-ins for Alembic_In and ScenegraphXml_In. To learn more on its use, see [Using the Importomatic](#).

- **ScenegraphGeneratorSetup** and **ScenegraphGeneratorResolve**

These nodes are used to take advantage of Katana's Scene Graph Generator API. For a brief overview, see [Generating Scene Graph Data with a Plug-in](#). For a more comprehensive explanation, please refer to the *development documentation*.



NOTE: Your studio may use its own geometry format, complete with a custom node to bring that format into Katana.

Adopting Alembic

Alembic is an open source scene information interchange framework. Alembic distills complex, animated scenes into non-procedural, application-independent, baked geometric results. It stores only the baked information and not how that information was obtained. For instance, a fully rigged and animated character would have its vertices efficiently stored for each frame of the animation but the control rig itself would not be stored. You can export to Alembic from most popular 3D applications.

For more information on Alembic, see <http://code.google.com/p/alembic/>.

Adding an Alembic Asset

To add an Alembic asset:

1. Create an Alembic_In node and add it to your recipe (assets are usually added first to any recipe).
2. Select the Alembic_In node and press **Alt+E**.

The Alembic_In node becomes editable within the **Parameters** tab.

3. In the **name** parameter, enter the scene graph location to place the Alembic data.
4. Enter the asset filename in the **abcAsset** parameter.

Describing an Asset Using XML

XML is a simple way to describe a hierarchical structure. Katana leverages this format to provide a rich descriptive asset language. Through XML, assets can be structured so they are loaded and manipulated in stages. Simpler versions of the asset (proxies) load quicker and use less memory, allowing you to only load the full asset when absolutely necessary.

Some asset elements that you can describe within a ScenegraphXml file are:

- Assembly locations
- Component locations
- Level-of-detail group locations
- Level-of-detail locations
- Other XML locations
- Geometry caches

Adding an Asset Described Using XML

To add an asset described using XML:

1. Create a ScenegraphXml_In node and add it to your recipe (assets are usually added first to any recipe).
2. Select the ScenegraphXml_In node and press **Alt+E**.
The ScenegraphXml_In node becomes editable within the **Parameters** tab.
3. In the **name** parameter, enter the scene graph location to place the data.
4. Enter the asset filename in the **asset** parameter.



NOTE: Providing a full description of how to describe a scene using XML is beyond the scope of the User Guide. For more information, consult the *developer's documentation* accessed through the **Help > Documentation** menu.

Using the Importomatic

The Importomatic node is used to bring in multiple assets and - if needed - assign them a look file or attribute file. Packaging this into one node keeps the recipe simpler to understand and debug.

With the Importomatic node you can:

- Read in multiple Alembic and ScenegraphXML assets in a single node.
- Assign look files to any of the assets (for more on look files, see [Look Development with Look Files](#)).
- Assign attribute files to any of the assets.
- Branch from the Importomatic node, allowing multiple outputs.

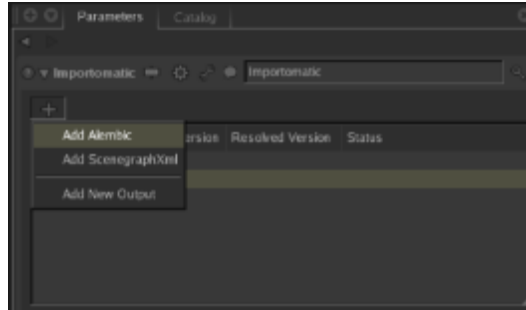


TIP: Providing a full description of how to describe a scene using XML is beyond the scope of the User Guide. For more information, consult the *developer's documentation* accessed through the **Help > Documentation** menu.

Adding Assets Using the Importomatic

To add assets using the Importomatic:

1. Create the Importomatic node and place it within the project.
2. Select it and press **Alt+E**.
The Importomatic node becomes editable within the **Parameters** tab.
3. Click **+** within the **Parameters** tab.
The asset and output menu is displayed.



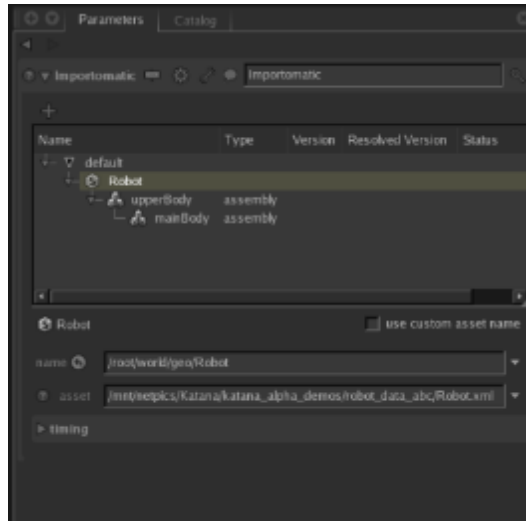
4. Select **Add Alembic** or **Add ScenographXml** and select the asset from the file browser or asset management browser.

The new asset is added to the Importomatic's hierarchy.

Editing an Importomatic Asset's Parameters

To edit an asset's parameters:

1. Select the asset within the Importomatic's hierarchy within the **Parameters** tab.
The asset's parameters are displayed below the hierarchy.
2. Make any changes to the asset that are needed. The parameters that are available are dependent on the asset type.

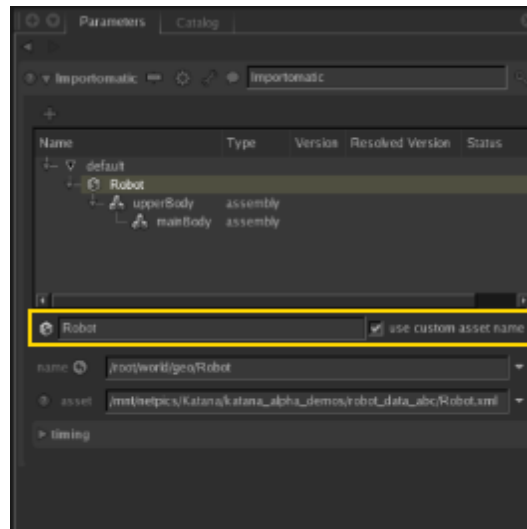


Editing an Asset's Name

To edit the name:

1. Select the asset within the Importomatic's hierarchy.
The asset's parameters are displayed below the hierarchy.
2. Toggle **use custom asset name** on.

The asset name becomes editable.



3. Change the asset name in the field directly below the hierarchy.
Changing the asset's name within the Importomatic does not influence its location within the **Scene Graph** tab.

Disabling an Asset

To disable an asset:

1. In the Importomatic parameters, right-click on the asset name within the hierarchy.
2. Select **Ignore Asset** (or press **D**).
The asset is no longer created.

Enabling an Asset

To enable an asset:

1. In the Importomatic parameters, right-click on the asset name within the hierarchy.
2. Select **Unignore Asset** (or press **D**).

Deleting an Asset from the Importomatic

To delete an asset:

1. In the Importomatic parameters, right-click on the asset name within the hierarchy.
2. Select **Remove Item** (or press **Delete**).

Assigning a Look File to an Asset

To assign a look file:

1. In the Importomatic parameters, right-click on the asset name within the hierarchy.
2. Select **Assign Look File**.
The file browser or your studio's asset picker displays.
3. Select the look file from the file browser or asset picker.

Assigning an Attributes File to an Asset

To assign an attributes file to an asset:

1. In the Importomatic parameters, right-click on the asset name within the hierarchy.
2. Select **Assign Attribute File**.
The file browser or your studio's asset picker displays.
3. Select the attribute file from the asset picker or file browser.



NOTE: Use attribute files to add attributes to existing locations. For a full explanation on using attribute files, see the accompanying PDF, which is accessed through the **Help > Documentation**.

Adding Additional Outputs to the Importomatic

To add an additional output:

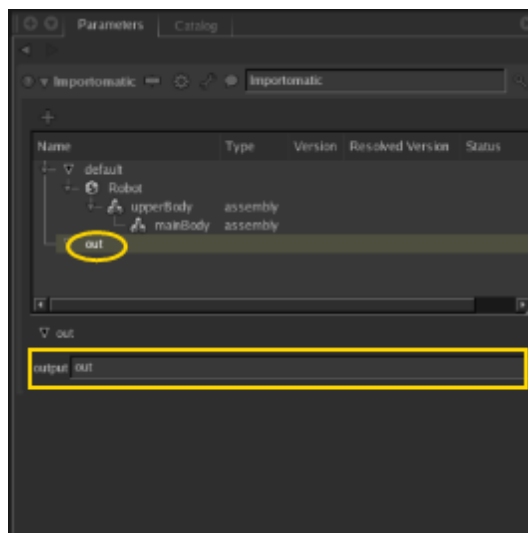
1. In the Importomatic parameters, click **+**.
The asset and output menu is displayed.
2. Select **Add New Output**.
A new output is added to the node and hierarchy.

Changing an Output's Name

Apart from the default output, the outputs from the Importomatic can be changed.

To change the name of an output:

1. In the Importomatic parameters, select the output in the hierarchy.
2. Type the new output name in the **output** parameter.



Generating Scene Graph Data with a Plug-in

Using the Scene Graph Generator (SGG) nodes, `ScenegraphGeneratorSetup` and `ScenegraphGeneratorResolve`, you can write a plug-in to generate locations and attributes. The coding of SGGs is explained within the *developer documents* that accompany the installation. To access the documentation, select **Help > Documentation**.

The first node of the pairing is `ScenegraphGeneratorSetup`. You can use this node to place the arguments needed for the plug-in into the scene graph. The procedure itself is not executed until either the recipe reaches the second node in the pair - the `ScenegraphGeneratorResolve` - or the renderer requests it at render time.

You can tag `ScenegraphGeneratorSetup` nodes with an ID, or with multiple IDs in a comma- or space delimited list. Subsequently, you can set a `ScenegraphGeneratorResolve` node to only resolve `ScenegraphGeneratorSetup` nodes with specified IDs.



NOTE: Acceptable characters for use in `ScenegraphGeneratorNode` IDs are upper and lower case letters, numbers, and spaces or commas to delimit lists.

Adding a Scene Graph Generator Location to Your Scene Graph

To add a Scene Graph Generator (SGG) location to your scene graph:

1. Create a `ScenegraphGeneratorSetup` node and place it within your recipe.
2. Select the `ScenegraphGeneratorSetup` node, and press **Alt+E**.
The `ScenegraphGeneratorSetup` node becomes editable within the **Parameters** tab.
3. If desired, give the node an ID by entering one in the **resolveIds** field. To give the node multiple IDs, enter a comma- or space delimited list.

4. Select the SGG from the **generatorType** dropdown.

The arguments for the SGG display.

The ScenegraphGeneratorSetup node creates a location of type **scenegraph Generator**. The attributes at that location include the **scenegraphGenerator.generatorType**, which is the procedure that is loaded, and **scenegraphGenerator.args**, which contains all the arguments needed by the procedures.

Forcing the Scene Graph Generator to Execute

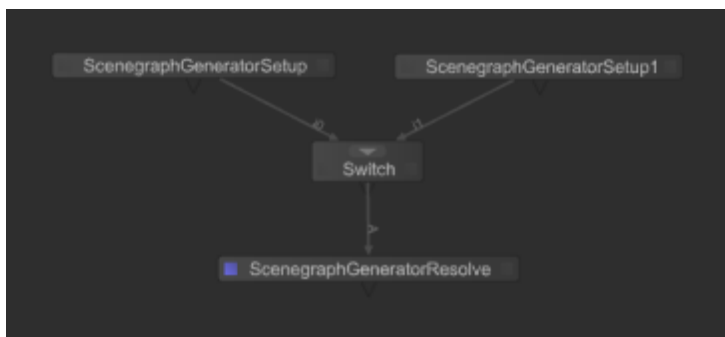
To force the SGG to execute:

1. Create a ScenegraphGeneratorResolve node and place it downstream of the ScenegraphGeneratorSetup node, at the point you want the ScenegraphGeneratorSetup to execute.
2. If you want to ignore ScenegraphGeneratorSetup node IDs, and resolve all ScenegraphGeneratorSetup nodes in the recipe, set the **resolveWithIds** field to **all** using the dropdown menu
3. If you want to resolve only ScenegraphGeneratorSetup nodes with specified IDs, set the **resolveWithIds** field to **specified** using the dropdown menu. Enter the IDs of the nodes to resolve into **specifiedResolveIds** field.



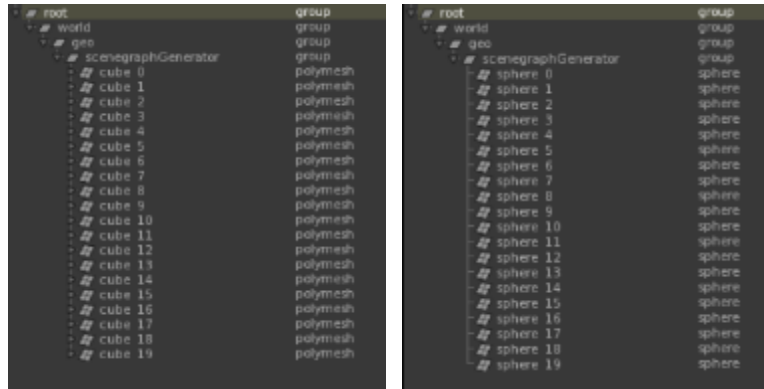
NOTE: Anything below the **scene graph generator** location at the time the generator is resolved is deleted.

To resolve a subset of all ScenegraphGeneratorSetup nodes in a recipe either use **resolveIds**, or pass the output of your ScenegraphGeneratorSetup nodes through a Switch node.



Passing through a Switch node allows you to have multiple ScenegraphGeneratorSetup nodes with the same name, and same scene graph location. Only one of them is passed downstream by the Switch, so only one downstream scene graph location is created.

The recipe pictured above has two ScenegraphGeneratorSetup nodes, one set to spheres, the other set to cubes. Only one scenegraphGenerator location is created, depending on the Switch setting. In this case, the scenegraphGenerator location is the result of either a ScenegraphGeneratorSetup node set to **SphereMaker**, or a ScenegraphGeneratorSetup node set to **CubeMaker**.



NOTE: You can give multiple ScenegraphGeneratorSetup nodes the same ID. Setting a ScenegraphGeneratorResolve node to **specified**, and entering an ID resolves all ScenegraphGeneratorSetup nodes with that ID.

A ScenegraphGeneratorSetup node with multiple IDs is resolved at any downstream ScenegraphGeneratorResolve node with **resolveWithIds** set to **specified** and any of the ScenegraphGeneratorSetup nodes IDs listed in the **specifiedPathResolveIds** field (or at any downstream ScenegraphGeneratorResolve node with **resolveWithIds** set to **all**).

Adding and Assigning Materials

Overview

A material is a scene graph location that holds one or more shaders. **Shaders** define how an object, such as a piece of geometry or a light, or - in the case of an Atmosphere shader - a volume, interacts within a renderer to create an image.

The most common types of materials are:

- **light materials** (complete with a light shader), which are assigned to light locations to illuminate a scene, and
- **geometry materials** (with surface shaders and possibly displacement or bump shaders), which are assigned to 3D geometry and particles.

The process of creating a basic material is broken down into two stages (although this can be done with one node):

1. Create the scene graph material location to hold the shaders.
2. Add the shaders to that location.

You can assign one material to multiple lights or pieces of geometry. To define this relationship between a material and its objects, use a MaterialAssign node.

An object with a material assigned keeps a reference to its material on the **materialAssign** attribute. The material is actually copied to the object's location either at render time, or at a MaterialResolve node.

At render time, a number of resolvers are applied automatically. These resolvers perform just-in-time resolving of certain operations that are usually best done at the last minute. These resolvers are called implicit resolvers. This method allows data to remain at a higher level for longer. For more details, see [Turning on Implicit Resolvers](#).



NOTE: Katana is a renderer agnostic application, and the shader types available depend upon the renderer plug-ins and how they locate their shader libraries.

Creating a Material

The first stage in creating a material is the creation of that material's location. This is the scene graph location that acts as a container for one or more shaders.

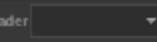
To create a material location:

1. Create a Material node and add it to your recipe.
Materials are usually created in their own branch and a Merge node is used to connect them to the rest of the recipe. If you need multiple materials, use a MaterialStack node. See [Multiple Materials with a MaterialStack Node](#).
2. Select the Material node and press **Alt+E**.
The Material node becomes editable within the **Parameters** tab.
3. Enter the material's name in the **name** parameter.
Although strictly not needed as Katana handles name clashes gracefully, it is good practice to name the material, as the name is used for both the node name and the material's scene graph location.
4. In the **namespace** parameter, enter the location below **/root/materials** to place the material.
By default, the material is placed below **/root/materials** in the scene graph. If **namespace** is not blank, the material is placed below **/root/materials/<namespace>**. Some of the most common namespaces are included as a dropdown to the right of the parameter. You can also specify nested namespaces, for instance, if the **namespace** parameter is **geo/metals**, the material is placed in the scene graph below **/root/materials/geo/metals**.

Adding a Shader to a Material Location

A material location needs to have one or more shaders attached.

To add shaders to the material location:

1. Follow steps 1 to 4 in [Creating a Material](#) above to create a material location.
2. Click **Add shader** and select a shader type.
The list of shader types varies depending on the renderers installed.
3. Add a shader to the new shader type's parameter. You can:
 - Click  to the immediate right of the shader type and select the shader from the list.

OR

 - Browse for a shader with  > **Browse...** and navigate to the shader using the **Shader Browser** dialog, select it and click **Accept**.
4. If you want to set any of the shader's parameters to non-default values, expand the parameters for the shader by clicking  and enter the changes where needed.

- Repeat steps 2 to 4 for any additional shaders for this material.

A possible combination might be a surface shader and a displacement shader. Material locations can have shaders from more than one renderer, only shaders for the appropriate renderer are selected at render time. This makes it possible for a single material to control how an object looks in a number of different renderers.

Creating a Material from a Look File

Materials previously baked out into Katana look files can also be assigned to material locations. Look files and the look development process is explained in greater detail in [Look Development with Look Files](#).



NOTE: This is different from reading in all the materials from a Katana look file, such as a material palette look file created during look development. Material palettes and their creation is covered in [Using Look Files to Create a Material Palette](#).

To use a material from a look file at this material location:

- Follow steps 1 to 4 in [Creating a Material](#) above to create a material location.
- Select **create from Look File** in the **action** parameter dropdown.
- Enter the path to the look file in the **lookfile** parameter, or click ▼ > **Browse...**, navigate to the look file and click **Accept**.
- Select a material from the **materialPath** dropdown list.
This is the list of materials contained within the look file. The list is automatically populated when a valid look file is assigned to the **lookfile** parameter.
- If you don't want to import the material as a reference, select **No** for the **asReference** parameter dropdown.
When Katana imports the material by reference, a reference to the original location of the material is kept. This enables any changes to the original material to be propagated downstream, even if this material is itself baked as part of a look file.
- If you need to change any parameters, expand the parameters for the shader(s) by clicking  and entering the changes where needed.

Creating a Material that is a Child of Another Material

A child material inherits all the shaders from the parent, but changes you make to the child do not influence the parent.

To create a child material:

- Follow steps 1 to 4 in [Creating a Material](#) above to create a material location.
- Select **create child material** in the **action** parameter dropdown.

3. Enter the scene graph location of the parent material in the **location** parameter within the **inheritsFrom** parameter grouping. See [Manipulating a Scene Graph Location Parameter](#) for details on scene graph location parameter fields.

The child material now has the same attribute values as the parent.

You can make any changes needed to the parameters in this node without changing the parent. This includes adding additional shaders.

Editing a Material

Once a material is created, it is not locked down. Later in the recipe, you can edit the material using another Material node.

To edit a material location:

1. Create a Material node and connect it to the recipe downstream of the target material.
2. Select the Material node and press **Alt+E**.
The Material node becomes editable within the **Parameters** tab.
3. Select **edit material** in the **action** parameter dropdown.
4. Enter the scene graph location of the material to edit in the **location** parameter within the **edit** parameter grouping. See [Manipulating a Scene Graph Location Parameter](#) for details on scene graph location parameter fields.
The shaders and their current parameter values are displayed below.
5. Edit the shaders for that material location wherever needed. This includes adding additional shaders.

Overriding a Material

As a material location can be assigned to multiple pieces of geometry, sometimes a geometry-specific change is needed. One way to perform this change is to use a material override. You point the Material node at the location(s) to override. Then any changes made are stored on the **materialOverride** attribute of the location.

It is also possible to override material locations directly. In this case, the override acts in the same way as an edit.

You can also override multiple materials at once, but only edit one.

To override the material at a geometry location:

1. Create a Material node and connect it to the recipe downstream of the target material.
2. Select the Material node and press **Alt+E**.
The Material node becomes editable within the **Parameters** tab.
3. Set the **action** dropdown to **override materials**.

4. Assign the scene graph locations of the geometry locations to the **CEL** parameter (located in the **overrides** parameter grouping). See [Assigning Locations to a CEL Parameter](#) for more on using **CEL** parameter fields.
5. In the **Scene Graph** tab, select the material location of the material assigned at the geometry location (or select  > **Select In Scene graph** on the **materialAssign** attribute of the geometry location).
6. Middle-click and drag from the attribute to override to the **Drop Attributes Here** hotspot at the top of the **attrs** parameter grouping.
 The attribute displays within the **attrs** parameter grouping and can now be overridden inside the **Parameters** tab.
 All changes you make are added as attributes to the location(s) specified by the **CEL** parameter (under the **materialOverride** attribute).

Multiple Materials with a MaterialStack Node

Having a chain of Material nodes would soon clutter up a recipe. To create multiple materials within one node, use the MaterialStack node.

Adding a Material

To add a material inside the MaterialStack node:

1. Select **Add > Add Material**.
 A new material is added to the **Add** list.
2. Enter a new name in the **name** parameter.
3. Follow steps 2 to 5 in [Adding a Shader to a Material Location](#).

Adding a Material from a Look File

To add a material from a look file inside the MaterialStack node:

1. Select **Add > Add Look File Material**.
 A new material is added to the **Add** list.
2. Enter a new name in the **name** parameter.
3. Follow steps 3 to 6 in [Creating a Material from a Look File](#).

Adding a Material as a Child

To add a material as a child of an existing material:

1. Select a material in the **Add** list.

2. Select **Add > Add Child Material**.

A new material is added below the selected material.

3. Enter a new name in the **name** parameter.
4. Make any changes needed to the parameters, you can also add additional shaders.



NOTE: The parent has to be within the MaterialStack node, otherwise the menu options are not available.

Duplicating a Material

To duplicate a material within the MaterialStack node:

Select the material node in the **Add** list, right-click and select **Duplicate Material**.

Disabling a Material

To disable a material within the MaterialStack node:

Select the material node in the **Add** list, right-click and select **Ignore Material** (or press **D**).

Deleting a Material

To delete a material from the MaterialStack node:

Select the material node in the **Add** list, right-click and select **Delete Material** (or press **Delete**).

Adding a Material Node from the Node Graph

To add Material nodes from the **Node Graph** into the MaterialStack node:

Shift+middle-click and drag the nodes into the **Add** list.

Moving Materials Within the Add List

To move materials within the **Add** list:

Middle-click and drag.

Incorporating PRMan Co-Shaders

RenderMan co-shaders enable shader components to be connected together and plugged into the parameter(s) of one or more shaders. This modular form of shader writing is very powerful.

In order for a shader to make use of a co-shader, the co-shader has to be defined higher (or in the same location) in the scene graph hierarchy. For instance, for a material assigned to **/root/world/geo/Robot/leg** to use a co-shader, that co-shader must be assigned to one of:

- **/root/world/geo/Robot/leg**
- **/root/world/geo/Robot**
- **/root/world/geo**
- **/root/world**
- **/root**

Setting Up a Co-Shader Material Location:

1. Follow steps 1 to 4 in [Creating a Material](#) to create a material location.
2. Click **Add shader** and select **coshader** (under the **prman** heading).
A new **prmanCoshaders** parameter grouping is displayed with a co-shader sub-parameter grouping called **shader**.
3. To the right of the **shader** sub-parameter grouping, select  > **Rename...** .
The **Rename Parameter** dialog displays.
4. Enter a new name for the co-shader and click **Accept** (or press **Return**).
5. Add a co-shader to the **shader** parameter. You can:
 - Click  in the large dropdown and select a co-shader from the list.

OR

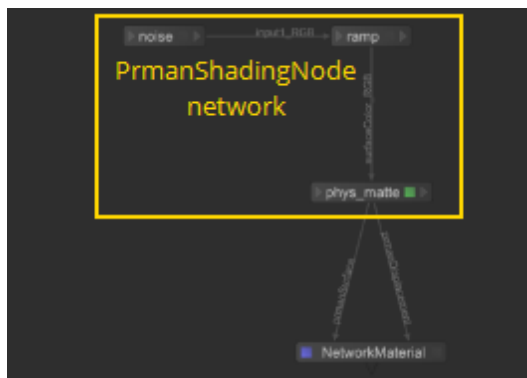
- Browse for a co-shader with  > **Browse...** and navigate to the co-shader using the **Shader Browser** dialog, select it and click **Accept**.
6. Edit the default parameters where needed.
 7. Repeat steps 2 to 6 if additional co-shaders are needed.
- You can create multiple co-shaders in the same node.



NOTE: You reference the co-shaders inside the main shader by the name selected in step 4 above.

Network Materials

Building a shader from a number of smaller parts is versatile and often efficient. Complicated shading networks can be built from simple reusable utility nodes. If a renderer supports the ability to build a shader in this manner, Katana provides the mechanism for connecting the output from one shader to the input of another. These shaders are connected using a renderer specific shading node, for instance a `PrmanShadingNode` node.



Network materials are connected into the recipe through the `NetworkMaterial` node. This creates a scene graph location and from this, you add terminals (also known as ports) depending on the type of shader you are creating, such as a PRMan surface shader. Just like a normal Material node, multiple types of shaders can be assigned to a single scene graph location, for instance a PRMan displacement shader can be connected to the same `NetworkMaterial` node as a PRMan surface shader.

Creating a Network Material

To create a network material, you first need to create a `NetworkMaterial` node:

1. Create a `NetworkMaterial` node and add it to your recipe.
Network materials are usually created in their own branch and a Merge node is used to connect them to the rest of the recipe.
2. Select the `NetworkMaterial` node and press **Alt+E**.
The `NetworkMaterial` node becomes editable within the **Parameters** tab.
3. Enter the material's name in the **name** parameter.
Although not strictly needed, as Katana handles name clashes gracefully, it is good practice to name the network material, as the name is used for both the node name and the material's scene graph location.
4. In the **namespace** parameter, enter the location below **/root/materials** to place the material.
By default, the material is placed below **/root/materials** in the scene graph. If **namespace** is not blank, the material is placed below **/root/materials/<namespace>**. Some of the most common namespaces are included as a dropdown to the right of the parameter. You can also specify nested namespaces, for instance, if the

namespace parameter is **geo/metals**, the material is placed in the scene graph below **/root/materials/geo/metals**.

5. Add the shader specific ports to the network material (discussed in [Adding Ports to a NetworkMaterial Node](#) below).
6. Connect a renderer specific shading node into the network material (see [Connecting Into a NetworkMaterial Node](#)).
7. Connect one or more renderer specific shading nodes into a network to form the complete shader (see [Using a Network Shading Node](#)).
8. Optionally, build the network material's interface to make it more artist friendly further down the pipeline (see [Creating a Network Material's Public Interface](#)).

Adding Ports to a NetworkMaterial Node

On its own, a NetworkMaterial node only creates a scene graph location and it needs to have terminals/ports added to allow the connection of shading nodes.

To add ports:

Click **Add terminal** and select a port type from the terminal type dropdown.

Multiple ports can be added to the same NetworkMaterial node.

Connecting Into a NetworkMaterial Node

A shading node is connected into a NetworkMaterial node's input port. The type of shading node that connects is renderer specific, for instance the ArnoldShadingNode node. Also, the shader that is assigned to the shading node needs to be of the correct type for the renderer and the NetworkMaterial node's port.

As an example, when creating a PRMan surface shader as a network material, the shader node that connects to the NetworkMaterial node's **prmanSurface** port must be a valid surface shader (either of type surface or when using a class based shader, implement one of the expected methods for a surface shader).

Connection Logic Checking

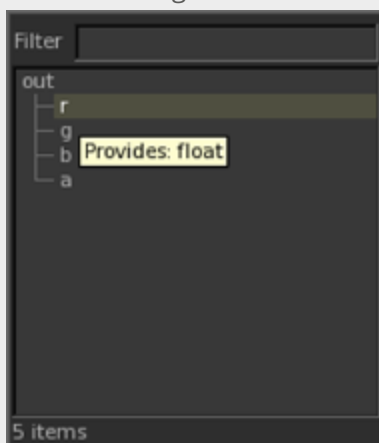
Connections between shading nodes support connection checking logic, so when connecting shading nodes together through the UI, only permissible connections are allowed. For example:

1. In an empty recipe, create two ArnoldShadingNodes. Set one to nodeType **image**, and the other to nodeType **spot_light**.
2. In the **image** type shading node, set the **filename** Parameter to point to an image.
3. Click on the right output arrow of the **image** shading node to see the available outputs, which are **r**, **g**, **b**, and **a**. Click on the **r** to select the red channel of the image shading node, then click on the left input arrow of the **spot_light** shading node.

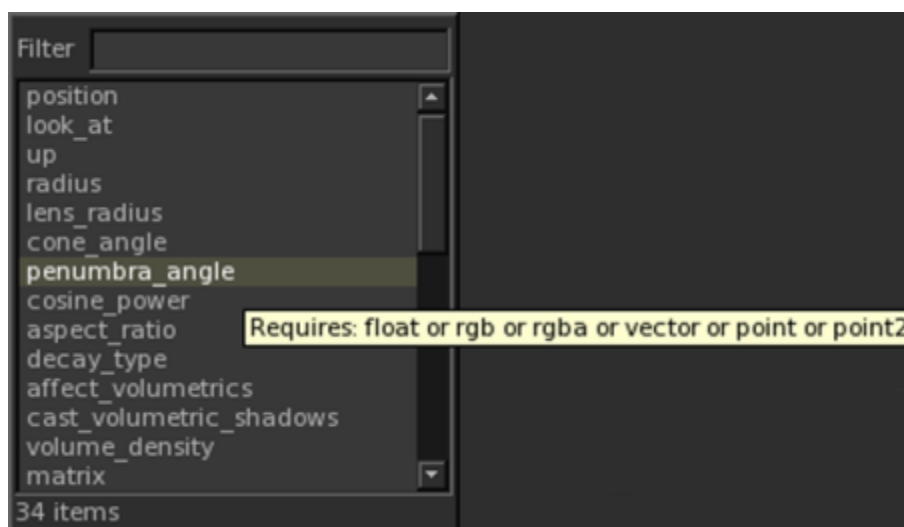
4. A new window shows the connection options. The input ports on the Standard node that cannot accept an **r** input - such as **decay_type** - are grayed out.



NOTE: Holding the mouse over an output, or input channel, shows the type it generates, or accepts.



5. The **r** channel output of the image nodes provides a float. Connect the **r** channel output of the image node to the **penumbra_angle** input of the spot_light node, which accepts float, RGB, RGBA, vector, point or point2 inputs.

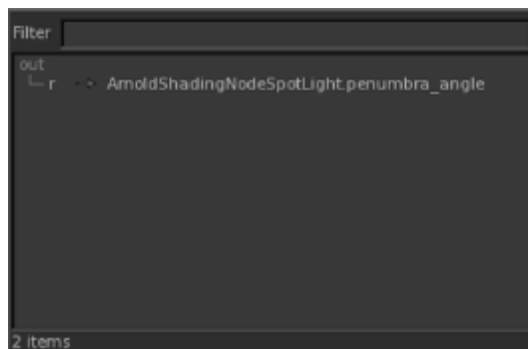


NOTE: To connect an output to an input, click on the output arrow of the source shading node, and click on the output you want. Then, click on the input arrow of the target shading node, and select the input you want to connect to.

Showing Connections

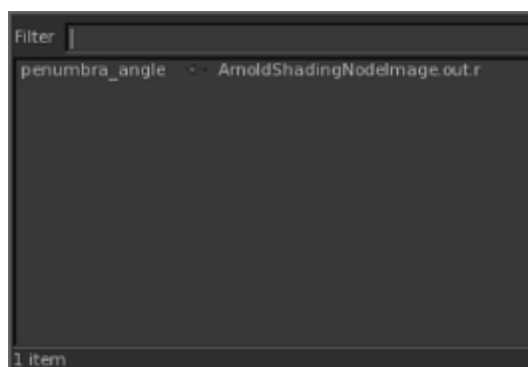
Once two shading nodes are connected, they're joined in the Node Graph by an arrow. Right-clicking on the arrow near the source node shows the outputs from that node, and what they connect to on the target node.

For example, open the recipe created in [Connection Logic Checking](#). Right-click on the arrow connecting the **image** and **spot_light** ArnoldShadingNodes. When the mouse is closer the image node, the following is displayed:



Indicating that the **r** output is connected to the **penumbra_angle** input of a node named - in this case - ArnoldShadingNodeSpotLight.

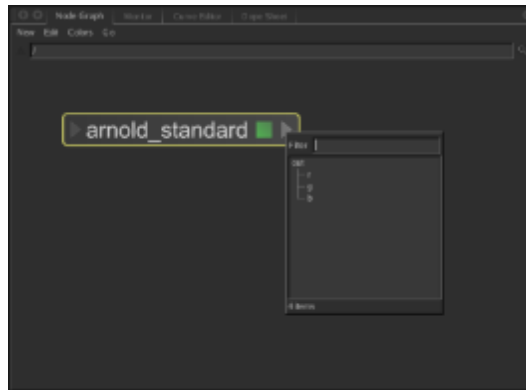
Right clicking with the mouse closer the spot_light shading node produces the following:



Indicating that the **penumbra_angle** input has an incoming connection from the **r** output of a node named - in this case - ArnoldShadingNodeImage.

Using a Network Shading Node

Shading nodes for network materials have a different appearance to other nodes.



Inputs for the shading node are accessed by clicking the triangle on the left of the node and outputs by clicking the triangle on the right. The green square shows when this node is editable in the **Parameters** tab. It is not possible to view the scene graph generated at this node. To view how this node influences the scene graph you can view the scene graph generated at its NetworkMaterial node.

Creating a Shading Node

1. Create a shading node and add it to the recipe.
The renderer name acts as a prefix for the shading node, so for PRMan the shading node is called PrmanShadingNode and for Arnold the shading node is ArnoldShadingNode.
2. Select the shading node and press **Alt+E**.
The shading node becomes editable within the **Parameters** tab.
3. From the **nodeType** dropdown, select the shader for this node.
The parameters for the shader display in the **parameters** grouping below **nodeType**.

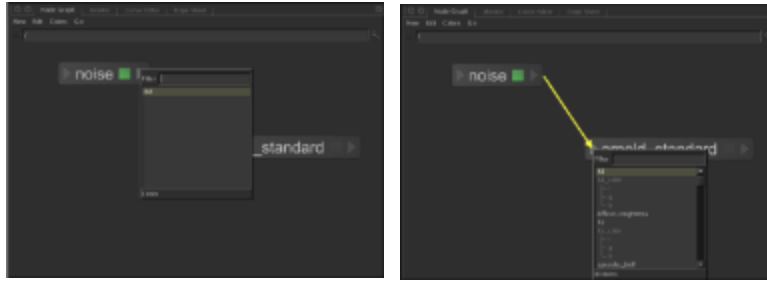


TIP: A quick way to name the node from the **nodeType** is to **middle**-mouse click and drag from the **nodeType** to the **name** parameter.

Connecting a Shading Node

There are two main ways to connect shading nodes, you can:

1. Click the right output triangle of the first shading node.
A list of possible outputs is displayed. The contents of the list depends on the shader.
2. Select an output from the list.
3. Click the left input triangle of the shading node with which to connect.
A list of possible inputs displays. Again, this list depends on the shader.
4. Select the input parameter from the list.



OR

1. Hover the cursor over the first node you want to connect.
2. Press the **Backtick** key (`) once.
3. Hover the cursor over the second node and press the **Backtick** key again.
The first output from the first node is connected to the first input of the second.

Using the first method above, it is also possible to connect the nodes in reverse order by first selecting the input parameter of one shading node and then selecting the output parameter of another.

Disconnecting a Shading Node

To disconnect one shading node from another:

1. Hover the mouse over the input connection and left-click when it turns yellow.
A connection list displays.
2. Select the link in the list.
The link becomes disconnected.
3. Click an empty area in the **Node Graph**.

Collecting Shading Nodes Inside a ShadingNodeSubnet

To help keep the shading network clearer, it is possible to group shading nodes inside a node, similar to a Group node, called a ShadingNodeSubnet. The main difference between a ShadingNodeSubnet node and a Group node is its ability to display and re-order the public interface (explained in [Creating a Network Material's Public Interface](#)) of the shading nodes within.

To add nodes to a ShadingNodeSubnet node:

1. Click the plus icon towards the bottom of the ShadingNodeSubnet to open the subnet's group.
2. Select the nodes to be added.
3. **Shift**+middle-click and drag the nodes over the opened subnet's group. When the subnet's group highlights, release the mouse button.

To add the shading node's public interface to the ShadingNodeSubnet's **Subnet Material Interface**:

1. Select the ShadingNodeSubnet node and press **Alt+E**.
The ShadingNodeSubnet node becomes editable within the **Parameters** tab and the public material interface is exposed.
2. Select the shading nodes with a public interface to expose and **Shift**+middle-click and drag the nodes into the **Subnet Material Interface** within the **Parameters** tab.

The public interfaces exposed in the **Subnet Material Interface** can be reordered, setting a preference for how they should be displayed downstream. This preference can always be overridden by the NetworkMaterial node itself, and only acts as a default.

To re-order the public interfaces, middle-click and drag one group or parameter to another valid location. An orange line highlights the new position. You can modify the order of the publicly-exposed parameters for a NetworkMaterial node by dragging and dropping them in the **Material Interface** widget, in the base NetworkMaterial node.

Creating a Network Material's Public Interface

As the network materials are sometimes built from a large number of shading nodes it might be difficult for an artist to work out which parameters are important. When building a network material, shading node parameters can be flagged as important, creating a public interface that is then exposed inside any network materials that use the shading node. These parameters are then used when editing the material or when using it to create a new material.

The public interface of a parameter can be nested using a page name, defined at the node level, and/or a group name, defined when exposing the parameter. When building the public interface any group name is appended to the end of a page name and any full stops are interpreted as the start of a sub-group. For instance:

- A page name of **image** with a group name of **coords** would place any parameters below **imagecoords**.
- A page name of **image**. with a group name of **coords** would place any parameters below **image > coords**.
- A page name of **image**. with a group name of **coords.s** would place any parameters below **image > coords > s**.
- An empty page name with a group name of **image.coords.s** would place any parameters below **image > coords > s**.



NOTE: To display the Network Material material interface parameters in the GafferThree object table's columns, see [Displaying Network Material Parameter Values](#)

Exposing a Shading Node's Parameters

1. Select the shading node and press **Alt+E**.
The shading node becomes editable within the **Parameters** tab and the **Subnet Material Interface** is exposed.
2. Click  next to the parameter to expose and select **Edit Parameter Name in Material Interface...**.
The **Material Interface Options** dialog displays.
3. Enter the details for the public interface in the dialog:

- In the **Name** field, enter the name for this parameter's public interface.
- In the **Group** field, enter the name for a group to parent this parameter's public interface under.
If only the **Group** field is populated, the parameter's public interface becomes the actual parameter name (grouped under the contents of **Group**).

Reordering the Parameters in the Network Material

The parameters with a public interface that are exposed in the network material's **Material Interface** can be reordered. The ShadingNodeSubnet node provides a hint as to the preferred order, but ultimately the order is decided by the network material. To reorder the interface of the network material in the **Material Interface**:

Middle-mouse click and drag the parameter or group.

Using the NetworkMaterialInterfaceControls Node

Logic can be applied to the public interface of a network material to change the visibility or lock status of pages or parameters. You can test parameter values using the following operators:

- **contains**
- **doesNotContain**
- **endsWith**
- **equalTo**
- **greaterThan**
- **greaterThanOrEqualTo**
- **in**
- **lessThan**
- **lessThanOrEqualTo**
- **notEqualTo**
- **notIn**
- **numChildrenEqualTo**
- **numChildrenGreaterThanOrEqualTo**
- **regex**

These tests can be combined using **and** as well as **or** logical operators.

What the node actually does is evaluate the test, for instance is the **samples** parameter **equalTo** 0, and use the result of that test to hide or lock the target interface element.

To make use of the NetworkMaterialInterfaceControls node:

1. Create a NetworkMaterialInterfaceControls node and add it to the recipe downstream of the NetworkMaterial node.

2. Select the `NetworkMaterialInterfaceControls` node and press **Alt+E**.
The `NetworkMaterialInterfaceControls` node becomes editable within the **Parameters** tab.
3. Add to the **materialLocation** parameter, the network material's scene graph location whose interface you want to control. For more on editing a scene graph location parameter, see [Manipulating a Scene Graph Location Parameter](#).
4. Select from the **state** parameter dropdown:
 - **visibility** - to have a page or parameter be visible based on the parameter test this node defines.
 - **lock** - to have a page or parameter locked based on the parameter test this node defines.
5. Select the type of interface element this node influences from the **targetType** parameter:
 - **page** (also referred to as a group)
 - **parameter**
6. In the **targetName** parameter, type the name of the network material's public interface element this node influences.
7. Select how the interface is controlled using the **definitionStyle** parameter dropdown. Selecting **operator tree** is assumed in the *Katana User Guide* as the **conditional state expression** option is beyond the scope of this document.
8. Select the type of test in the **op** parameter (under **operators** > **ops**).
9. Enter the name of the interface element in the **path** parameter to perform the test against.
10. Enter the value for the test in the **value** parameter.
11. Add any additional tests needed using the **Add** > ... menu.

Changing a Network Material's Connections

After a network material and its corresponding shading network has been built, the connections can be edited downstream through the use of the `NetworkMaterialSplice` node. This is especially useful when a network material has been read in from a look file and you need to edit the connections and/or make additions to the shading nodes.

To edit the connections after the shading network has been created, use the `NetworkMaterialSplice` node. There are two main ways to use it, to add in additional shading nodes, or to change the connections that exist inside the current network material. You can do both operations with the same node.

Appending New Shading Nodes to an Existing Network Material

1. Create a `NetworkMaterialSplice` node and connect the **in** port to the part of the recipe that contains the network material.
2. Add to the **location** parameter, the network material's scene graph location to append. For more on editing a scene graph location parameter, see [Manipulating a Scene Graph Location Parameter](#).
3. Connect the shading network to append to the append port of the `NetworkMaterialSplice` node.
4. Under **inputs** > **append**, click .

A dialog with the current network material's shading network displays.

5. In the dialog, select the input to connect into by clicking the left arrow of a shading node and selecting the appropriate input.

Adding Extra Connections for the Shading Nodes in a Network Material

1. Create a NetworkMaterialSplice node and connect the **in** port to the part of the recipe that contains the network material.
2. Add to the **location** parameter the network material's scene graph location. For more on editing a scene graph location parameter, see [Manipulating a Scene Graph Location Parameter](#).
3. To the right of the **extraConnections** parameter grouping, click **Add > Add Entry**.
A new parameter group is added, the first one is called **c0**, subsequent entries are incremented, such as **c1**, **c2**.
4. To the right of the **connectFromNode** parameter, click .
5. Select the output from one of the shading nodes, this is where the connection comes from.
6. To the right of the **connectToNode** parameter, click .
7. Select the input to one of the shading nodes, this is where the connection goes to.

Deleting Connections Between Shading Nodes in a Network Material

1. Create a NetworkMaterialSplice node and connect the **in** port to the part of the recipe that contains the network material.
2. Add to the **location** parameter the network material's scene graph location. For more on editing a scene graph location parameter, see [Manipulating a Scene Graph Location Parameter](#).
3. To the right of the **disconnections** parameter grouping, click **Add > Add Entry**.
A new parameter group is added, the first one is called **d0**, subsequent entries are incremented, such as **d1**, **d2**.
4. To the right of the **node** parameter, click .
5. Select the input to one of the shading nodes, this is the connection that is disconnected.

Editing a Network Material

You can edit the attributes created by the network material and shading network in two ways, either using the interface previously created or more directly, edit the attributes of the original shading nodes.

Editing the Network Material Using Its Interface

To edit a network material using its interface, you need to edit the scene graph location, in the same way as a normal material, using the Material node. To do this just, follow the steps in [Editing a Material](#)

It's necessary to use a Material node, even for a network material, and the node must be able to accept edits. In addition, the **makeInteractive** option on the Material node needs to be set to **yes** in order for the edits to be made interactively in the Viewer.

Editing the Shading Node's Attributes

The shading node's parameters are stored as attributes on the network material. The interface is used to expose some of these parameters and attributes for easy editing. Sometimes you may need to edit the attributes not exposed. Editing the attributes that aren't exposed is done with the NetworkMaterialParameterEdit node.

To perform an edit with the NetworkMaterialParameterEdit node:

1. Create a NetworkMaterialParameterEdit node and connect it to the recipe at the point you want to make an edit.
2. Select the NetworkMaterialParameterEdit node and press **Alt+E**.
The NetworkMaterialParameterEdit node becomes editable within the **Parameters** tab.
3. Add to the **location** parameter the network material's scene graph location to edit. For more on editing a scene graph location parameter, see [Manipulating a Scene Graph Location Parameter](#).
4. Select any shading nodes to edit by either:
 - selecting **Add > <shading node name>**, or
 - clicking  and right-clicking on any shading nodes to edit and selecting **Expose Parameters**.
5. Make any changes to the shading nodes inside the **nodes** parameter grouping.

Assigning Materials

As mentioned in the introduction, a material location needs to be associated with a geometry or light location. This is achieved with the MaterialAssign node.

To assign a material to a scene graph location:

1. Create a MaterialAssign node and connect it to the recipe after both the geometry and material locations have been created.
2. Select the MaterialAssign node and press **Alt+E**.
The MaterialAssign node becomes editable within the **Parameters** tab.
3. Add the scene graph locations where the material is to be assigned to the **CEL** parameter. See [Assigning Locations to a CEL Parameter](#) for more on using **CEL** parameter fields.
4. Enter the scene graph location of the material to assign in the **materialAssign** parameter. See [Manipulating a Scene Graph Location Parameter](#) for details on scene graph location parameter fields.



TIP: The best way to enter a material into the **materialAssign** parameter is to **Shift**+middle-click and drag from the Material node in the **Node Graph** tab to the **materialAssign** parameter. This creates an expression linking the material created by the Material node to the **materialAssign** parameter.

Assigning Textures

Shaders may be responsible for how a geometry location is rendered, but a lot of the time, the richness of the render comes from a number of asset-specific textures. These textures sometimes need to be passed to the shader on a per-asset basis.

Katana provides a number of ways to assign textures to assets depending on your pipeline. For an in-depth description of the options, please read the *Katana Technical Guide* which is found either:

- inside the Katana installation at: `${KATANA_ROOT}/docs/pdf/Katana_TechnicalGuide.pdf`, or
- inside Katana, select **Help** > **Technical Guide**.

Using Face Sets

When assigning materials to assets, it is often useful to break up the asset into smaller parts based on its faces. This allows different materials to be assigned to the different parts.

Creating a Face Set

Face sets are just a list of faces for a particular polymesh or subdivision surface. To create a face set:

1. Create a FaceSetCreate node and connect it to the recipe at the point you want to create the face set.
2. Select the FaceSetCreate node and press **Alt+E**.

The FaceSetCreate node becomes editable within the **Parameters** tab.

3. Add to the **meshLocation** parameter the polymesh or subdivision surface scene graph location. For more on editing a scene graph location parameter, see [Manipulating a Scene Graph Location Parameter](#).
4. Enter the name of this face set in the **faceSetName** parameter.

This is the name that is displayed in the **Scene Graph** tab below the **meshLocation** scene graph location.

5. Switch the **Viewer** tab into face set selection mode by:
 - selecting the polymesh or subdivision surface in the **Scene Graph** tab and clicking  in the **Viewer** tab, or
 - **Shift**+middle-click and drag the FaceSetCreate node onto the  icon, or
 - middle-click and drag the **selection** parameter name onto the  icon.
6. Select the faces for this face set. You can:
 - Select individual faces or marquee select multiple faces.
 - Use the **Shift** key while selecting to toggle whether a face is included, or the **Ctrl** key to remove faces, or hold **Ctrl+Shift** to add faces.

- Select **Selection** > **X-ray Selection** in the **Viewer** tab to toggle between only selecting the faces that are visible, and selecting all faces encompassed by the selection.

7. When you are happy with the selection, click ▼ next to the **selection** parameter and then select **Adopt Faces From Viewer**.

The **Viewer** tab exits face selection mode and the currently selected faces are copied to the **selection** parameter.



TIP: You can invert the selection using the **invertSelection** checkbox. Using two FaceSetCreate nodes, this feature, and an expression between the **selection** parameters, you can assign materials to both halves of an asset.

Editing a Face Set

If you need to edit an existing face set, you can:

- **Shift**+middle-click and drag the FaceSetCreate node onto the  icon, or
- middle-click and drag the **selection** parameter name onto the  icon.

This puts the **Viewer** tab into face selection mode and enables you to edit the faces selected following the steps from Step 6 in [Creating a Face Set](#).

Assigning Materials to a Face Set

Assigning materials to a face set is done in the same way as assigning a material to any other location. Using a MaterialAssign node to edit the **materialAssign** attribute of the face set's scene graph location.

Forcing Katana to Resolve a Material

By default, Katana connects a geometry or light location with its respective material using the location's **materialAssign** attribute. This attribute points to where the material is located within the scene graph. At render time, an implicit resolver copies the material, pointed to by the **materialAssign** attribute, to the geometry or light's location. For more on implicit resolvers and their benefits, see [Turning on Implicit Resolvers](#).

You can force material resolving at an earlier point within a recipe using the MaterialResolve node.

To force materials to be resolved earlier within the recipe:

Create a MaterialResolve node and connect it to the recipe at the point materials should be resolved.

Lighting Your Scene

Lights are light scene graph locations with a light material assigned. The light material contains a shader, which defines how that light illuminates the scene.

One strength of Katana is its ability to only load parts of the scene graph at render time if they are needed. Lights can potentially be anywhere within the scene graph hierarchy. Katana needs to keep a list of all lights so it doesn't need to traverse what could potentially be a very complicated scene graph, just to find all the lights. The light list is stored in the **lightList** attribute under the **/root/world** location.

Creating a Light in Katana

Creating a light inside Katana can be done in two ways. Using separate nodes (LightCreate and Material), or using the GafferThree node (or the legacy Gaffer node), which packages up light creation with a number of other useful functions.

To create a light from its core components:

1. Create a LightCreate node and place it within your recipe.
2. Create a Material node and connect the output of the LightCreate node to the input of the Material node.
3. Select the Material node and press **Alt+E**.

The Material node becomes editable within the **Parameters** tab.

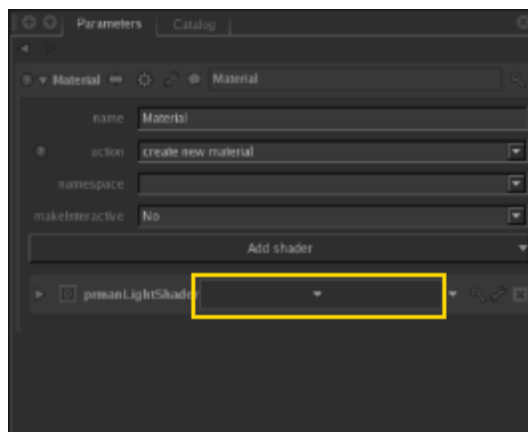
4. Select **Add shader > prman > light** within the Material node's **Parameters** tab.

A new PRMan light shader is added to the Material node.

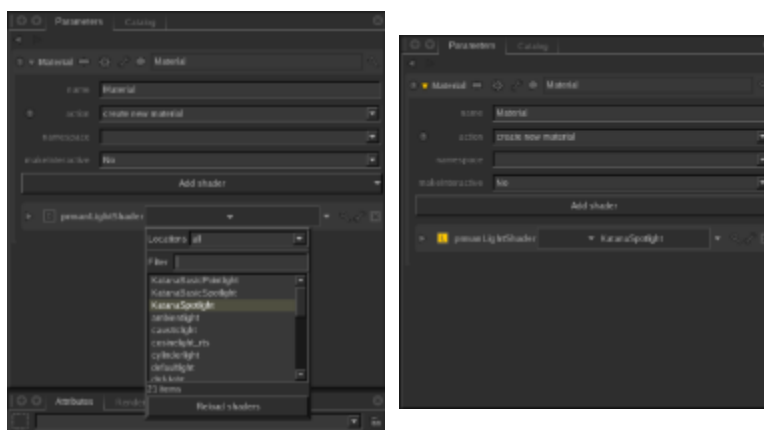


NOTE: The shader you select in the **Add Shader** dropdown doesn't necessarily need to be **prman**. Depending on your studio's setup, you may wish to select a different shader, and this can impact which light options you choose. For the purpose of these instructions, the **prman** shader has been chosen.

5. Click the arrow to the right of **prmanLightShader** to display the shader options.

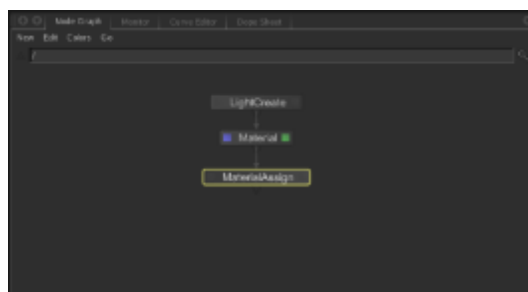


6. Select the type **KatanaSpotlight** from the dropdown.
The light name is populated in the dropdown button.



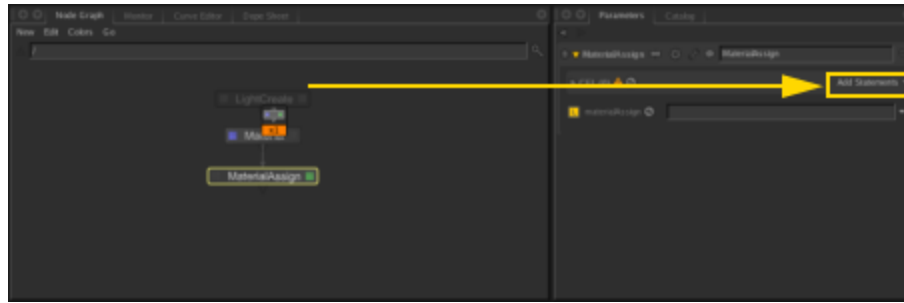
NOTE: **KatanaSpotlight** is a simple PRMan light shader that ships with Katana. Depending on your studio's setup, you may need to choose another light shader.

7. Create a MaterialAssign node and connect it to the output of the Material node.



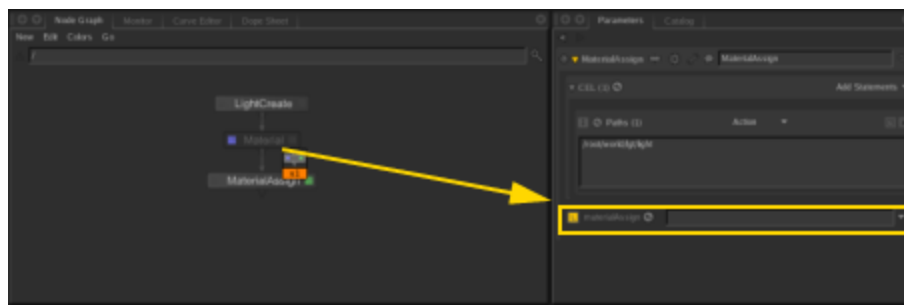
8. Select the MaterialAssign node and press **Alt+E**.
The MaterialAssign node becomes editable within the **Parameters** tab.
9. **Shift**+middle-click and drag from the LightCreate node in the **Node Graph** tab to the **Add Statements** dropdown in the **Parameters** tab.

The **Parameter Expression** field is populated with the scene graph location.



10. **Shift**+middle-click and drag from the Material node in the **Node Graph** tab to the **materialAssign** field in the **Parameters** tab.

An expression is created for the **materialAssign** parameter that evaluates to the location created by the Material node.



Using the MaterialAssign node, the PRMan light shader defined in the Material node is now assigned to the light defined in the LightCreate node.

Getting to Grips with the GafferThree Node

The method described in [Creating a Light in Katana](#), although valid, would be slow for a large number of lights. Katana's GafferThree node wraps light creation into a single node and adds the ability to:

- Create more than one light.
- Add rigs to group lights together.
- Mute and solo lights and rigs.
- Link lights to specific objects.
- Add aim constraints to lights.



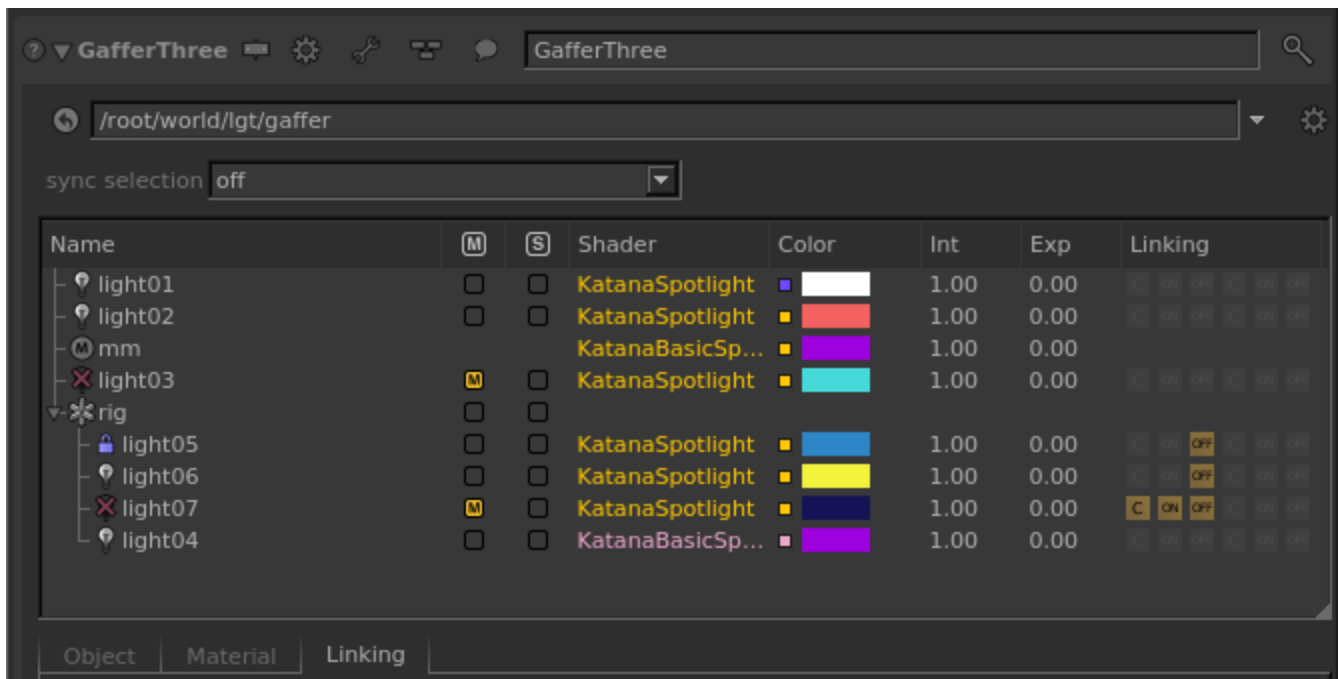
NOTE: Some of the options listed in [Creating a Light Using the GafferThree Node](#) may not be available due to the extensive customizability of Katana. Some of the GafferThree node's menu options are created using profiles, which can result in different light creation menu options.



NOTE: Using both Gaffer and GafferThree nodes together in a single node graph is not supported as it can result in unexpected mute and solo behavior.

Gaffer Object Table Overview

Once you've created a GafferThree node, use the Gaffer object table to manage the lights, rigs, and master materials within your scene.



Here are some tips on how to use the Gaffer object table:

- You can edit the parameters' values for multiple items at once. Simply select the items you need to edit and change the values directly in the Gaffer object table.
- The items are displayed in different colors depending on how their values have been set:
 - Gray/white: value set as "default"
 - Yellow: value set locally
 - Blue: value set as **forced default**
 - Pink: value inherited from a referenced master material
- Right-click a cell in the Gaffer object table to display a context menu with commands for manipulating underlying parameters. For instance, right-clicking in the **Shader** column allows you to add a renderer-specific shader. You can also define your own context menu for custom columns through the `createContextMenu()`.



NOTE: To display the Network Material material interface parameters in the GafferThree object table's columns, see [Displaying Network Material Parameter Values](#)

Creating a Light Using the GafferThree Node

To create a light with the GafferThree node, you need first to create a light and then add a light shader to that light. To add a light, do the following:

1. Create a GafferThree node and place it within the project.
2. Select the GafferThree node and press **Alt+E**.
The GafferThree node becomes editable within the **Parameters** tab.
3. In the GafferThree node's **Parameters** tab, right-click in the Gaffer object table and select **Add > Light** or press **L**.

A light is added in the Gaffer object table.



NOTE: The light, rig, and master material locations are created under **/root/world/lgt/gaffer** by default. You can change their scene graph locations by setting a new path in the root location field in the GafferThree node's **Parameters** tab.

4. To add a light shader to a light, you can add it through either the Gaffer object table or the **Material** tab in the **Parameters** tab.

To add a light shader through the Gaffer object table, follow the steps below:

1. Right-click in the **Shader** column, then select **Add Shader** and select a renderer-specific shader.
Nothing appears below the **Shader** heading.
2. Double-click below the **Shader** heading in-line with the light and select **KatanaSpotlight** from the list.
The name of the light shader, **KatanaSpotlight**, appears in the **Shader** column in the Gaffer object table and in the **Material** tab as well.

OR

To add a light shader through the **Material** tab in the **Parameters** tab, follow the steps below:

1. Select the newly added light in the Gaffer object table.
2. In the **Parameters** tab, click on the **Material** sub-tab .
3. Click on the **Add Shader** dropdown.
4. Select a renderer-specific shader from the list.
The renderer-specific shader appears in the **Material** tab.
5. Click on the newly added shader and select **KatanaSpotlight** from the list.

The name of the light shader, **KatanaSpotlight**, appears in the **Material** sub-tab and in the **Shader** column of the Gaffer object table as well.



NOTE: **KatanaSpotlight** is a simple PRMan light shader that ships with Katana. Depending on your studio's setup, you may need to choose another light shader.



TIP: Only shaders for the default renderer are displayed when double-clicking the **Shader** column. To assign shaders for a different renderer, right-click in the **Shader** column or use the **Material** tab below the Gaffer object table.

If you want to set or amend the default renderer for Katana, refer to [Changing the Default Renderer](#) for details on setting the DEFAULT_RENDERER environment variable. If this environment variable is not set, PRMan is assumed to be default.

Making Use of Rigs

Rigs create a scene graph group complete with transform attributes and the ability to easily add constraints. Lights created below the rig inherit its transformations, which enables you to move the lights around as one. Rigs can also be exported and imported.

Creating a Rig

In the GafferThree node's **Parameters** tab, right-click in the Gaffer object table and select **Add > Rig** or press **R**.

A rig is created in the Gaffer object table.



NOTE: You can also nest rigs. Simply right-click on an existing rig and select **Add > Rig** or press **R**.

Adding a Light to a Rig

In the Gaffer object table, right-click on a rig and select **Add > Light** or press **L**.

A light is created under the rig location.



NOTE: You can expand or collapse the lights nested under a rig location. Simply right-click on a rig and select the required option from the dropdown menu.

Importing a Rig

1. In the GafferThree node's **Parameters** tab, right-click a location within the Gaffer object table and select **Add > Import Rig....**

The **Import Rig** dialog displays.

2. Select the rig file in the dialog and click **Accept**.

The rig is imported under the selected location.

Exporting a Rig

1. Right-click on the rig to export and select **Export Rig**.

The **Export Rig** dialog displays.

2. Navigate within the dialog to where you wish to save the rig and enter a rig name.

3. Click **Accept**.

The rig is saved with a **.rig** file extension.

Adding a Point Constraint to a Rig

To add a point constraint:

1. Select the rig and click the **Parameters > Object** sub-tab.
2. Check **enable point constraint** and open the **point constraint options** parameter grouping.
3. Enter the scene graph location for the target in the **targetPath** parameter. For more on using scene graph location parameters, see [Manipulating a Scene Graph Location Parameter](#).
4. Click the **targetOrigin** dropdown and select the part of the target to use as the point constraint:
 - **Object** - the object's transform position is used.
 - **Bounding Box** - the center of the object's bounding box is used.
 - **Face Center Average** - the center of all the faces for the object are averaged to create the point constraint position.
 - **Face Bounding Box** - the center of the face's bounding box is used.

Adding an Orient Constraint to a Rig

To add an orient constraint:

1. Select the rig and click the **Parameters > Object** sub-tab.
2. Check **enable orient constraint** and open the **orient constraint options** parameter grouping.
3. Enter the scene graph location for the target in the **targetPath** parameter. For more on using scene graph location parameters, see [Manipulating a Scene Graph Location Parameter](#).

4. Select the axes to constrain (by default, it's all three). To remove the constraint for any of the individual axes, click the checkbox to disable it.

Defining a Master Light Material

At times it is best to have a master material and set local overrides per light. This can be done within the GafferThree node by creating a master material and assigning it to a light. Any changes made within the light's **Material** sub-tab act as an override for the master material.

Creating a Master Material

In the GafferThree node's **Parameters** tab, right-click in the Gaffer object table, and select **Add > Master Material** or press **Alt+M**.

A master material location is created inside the GafferThree node.

Assigning a Master Material to a Light

1. In the **Parameters** tab, double-click below the **Shader** heading in-line with the light you want to assign the master material to.
2. Select the master material from the list (master materials are displayed in pink).
The master material is assigned to the light.

Adding an Aim Constraint to a Light

Lights created inside the GafferThree node come with the ability to use an aim constraint. Using an aim constraint makes the light point at an object (the target) within the scene.

Adding an Aim Constraint

1. In the **Parameters** tab, select the light within the Gaffer object table.
2. In the **Object** sub-tab of the Gaffer object table, select the **enable aim constraint** checkbox.
The **aim constraint options** grouping displays.
3. In the **aim constraint options** grouping, enter the aim target in **targetPath** (for more details on how to enter a scene graph location, see [Manipulating a Scene Graph Location Parameter](#)).

Changing the Aim Constraint's Center Point

Select from the **targetOrigin** dropdown:

- **Object** - the point defined by the transform of the object.

- **Bounding Box** - the center of the bounding box.
- **Face Center Average** - the average from all the face centers.
- **Face Bounding Box** - the bounding box of all the faces.



NOTE: Using **Face Center Average** or **Face Bounding Box** could be slow for heavy geometry.

Linking Lights to Specific Objects

Light linking allows you to light a set of objects while others aren't or turn off the lights on a set of objects while the others are lit.



NOTE: By default the whole scene is lit. In order to light only one specific object, you need first to turn off all the lights for the entire scene (**/root/world/geo** location).

To link a light to a specific object in the scene, do the following:

1. Ensure your light is positioned towards the object you want to light.
2. In the Gaffer object table's **Parameters** tab, select a light and click on the **Linking** sub-tab.
The parameters of the **Linking** sub-tab are displayed.
3. Click on the **light linking** section.
The **on** and **off** CEL widgets are displayed.
4. **Shift**+middle-click and drag from the **/root/world/geo** location in the **Scene Graph** tab to the **off** CEL widget in the **Parameters > Linking** sub-tab.
All the lights are turned off in your scene. In the Gaffer object table, the icon "off" appears in the **Linking** column to indicate you have set up a light link of type "off".
5. **Shift**+middle-click and drag from the chosen object location in the **Scene Graph** tab to the **on** CEL widget in the **Parameters > Linking** sub-tab.
The light selected in the Gaffer object table lights the chosen object only. In the Gaffer object table, the icon "on" appears in the **Linking** column to indicate you have set up a light link of type "on".



WARNING: If a location is matched by both the **on** and **off** CEL expressions, then the **on** CEL expression overrides the **off** CEL expression.



NOTE: To light several objects, drag as many object locations as needed from the **Scene Graph** tab to the **on** CEL widget in the **Parameters > Linking** sub-tab.

Disabling a Light for Specific Object

To disable a light, follow the steps below:

1. In the Gaffer object table's **Parameters** tab, select the light you want to disable and click on the **Linking** sub-tab.
The parameters of the **Linking** sub-tab are displayed.
2. Click on the **light linking** section.
The **on** and **off** CEL widgets are displayed.
3. **Shift** +middle-click and drag from the chosen object location in the **Scene Graph** tab to the **off** CEL widget in the **Parameters > Linking** sub-tab.
The selected light for the chosen object location is disabled in your scene. In the Gaffer object table, the icon "off" appears in the **Linking** column to indicate you have set up a light link of type "off".



WARNING: If a location is matched by both the **on** and **off** CEL expressions, then the **on** CEL expression overrides the **off** CEL expression.



NOTE: For information on how to use the **clearUnmatched** parameter, see *Katana Reference Guide*, chapter *Nodes D-H*.

Linking Shadows to Specific Objects

Linking shadows is handled in the same manner as linking lights. Each location within the scene graph under **/root/world** has a **lightList** attribute. This is where light linking and shadow linking information is stored.

Adopting Items from Incoming Scene

You can adopt lights, rigs, and master materials from any upstream GafferThree nodes and then override any of their set parameters.

Adopting Items

To adopt items from the incoming scene, do the following:

1. Select the GafferThree node in which you want to collect the adopted items and press **Alt+E**.
The GafferThree node's parameters display in the **Parameters** tab.
2. In the GafferThree node's **Parameters** tab, click on the  button and select **Show Incoming Scene** from the dropdown menu.

All adopted lights, rigs and master materials, along with their parameters, are displayed in the Gaffer object table. The names of the items are displayed in italics showing they are adopted items. The light gray color indicates they are read-only items.



NOTE: The disabled lights, rigs, and master materials from any upstream nodes don't show when you adopt items from the incoming scene.

Editing Adopted Items

Any adopted item from the incoming scene is by default read-only

To make the adopted items editable, right-click on the name of the item you want to edit and select **Adopt for Editing** from the dropdown menu.

The name of the item displays in a white color showing it is editable. You can now make any changes to its parameters.



NOTE: You can edit the parameters of the adopted items and also add child lights, rigs and master materials to the adopted rigs.

Deleting Edits Made to Adopted Items

To delete edits made to the adopted items and retrieve their original values, in the Gaffer object table, right-click on an adopted item and select **Delete Edit Package** from the dropdown menu.

The parameters of the adopted item are set back to their original values (adopted items' parameters from the incoming scene) and the adopted item is now read-only.

Soloing and Muting Lights and Rigs

You can either solo or mute lights in a scene. Soloing a light means that you are only keeping that one specific light to light the scene. All the other lights are automatically set as "mute" and therefore turned off. Muting lights means you can turn specific lights off while others are lit. Soloing or muting rigs works the same way.

Soloing Lights and Rigs

To solo a light, do the following:

1. In the Gaffer object table, select the light you want to solo.
2. In the Solo column  check the box in-line with the selected light.
The light is set as "solo" and all the other lights are automatically set as "mute".

In the Mute column , the icon  shows which lights are set as "mute" and it is also displayed in the **Scene Graph** tab to indicate which light locations are in a mute state.



NOTE: Soloing lights with the GafferThree is persistent across sessions, which is not the case with the Gaffer.



NOTE: To solo a rig, follow the same procedure but select a rig instead of a light.



NOTE: Soloing a light in a GafferThree node affects any lights created in other downstream GafferThree nodes.

Muting Lights and Rigs

To mute a light, do the following:

1. In the Gaffer object table, select one or as many lights as you want to mute.
2. In the Mute column  check the box in-line with the selected light or one of the selected lights for multiple selection.

The selected lights are set as "mute" while all the other lights still light the scene.

In the Mute column , the icon  shows which lights are set as "mute" and it is also displayed in the **Scene Graph** tab to indicate which light locations are in a mute state.



NOTE: To mute one or more rigs, follow the same procedure but select rigs instead of lights.

Locking a Light or Rig's Transform

Once a light is in the correct position, you can lock it to prevent accidental movement. Locking a light does not prevent it from being edited or deleted.

To toggle whether a light is locked, right-click on the light and select **Toggle Lock State of Selected Items** or press **Ctrl+L**.

The blue lock  icon is either added or removed in place of the light icon in the Gaffer object table.



NOTE: Locking a rig works the same way. Simply right-click on the rig and select **Toggle Lock State of Selected Items** or press **Ctrl+L**.

Duplicating an Item Within the Gaffer Object Table

To duplicate an item within the Gaffer object table, right-click on the item and select **Duplicate**.

The item is duplicated into the Gaffer object table and a number is appended to the name.

Syncing the GafferThree Selection with the Scene Graph

When editing lights using both the GafferThree and manipulators in the Viewer, it is often convenient to sync the selection of those lights between the two locations. The sync selection parameter in the GafferThree node's parameters allows you to set sync selection in three ways:

- **off** - no syncing is performed (the default).
- **out** - selection of a light in the GafferThree node is mirrored in the **Scene Graph** tab, but not the other way around.
- **in/out** - selecting in either the **Scene Graph** tab or GafferThree node results in the corresponding entry in the other also being selected.

Deleting from the Gaffer Object Table

To delete an item from the Gaffer object table, right-click on the item you want to delete and select **Delete** or press the **Delete** keyboard shortcut.

Displaying Network Material Parameter Values

In order to show values of parameters from the public interface of a Network Material in specific columns of the GafferThree object table in the **Parameters** tab, you can set parameter meta name attributes in the **material** group attribute of the **material** location that defines the Network Material. To set these attributes, do the following:

- Use string attributes.
- Create names using the following syntax: `material.meta.[parameterMetaName].[rendererName]`
For example, **material.meta.exposure.arnold**
- Create values using the following syntax: `parameters.[parameterName]`
For example, **parameters.exposure**

You can set attributes one at a time using AttributeSet nodes, or you can set multiple attributes at once using OpScript nodes. Below is an OpScript code sample that can be used to create parameter meta name attributes for all parameters of a Network Material's interface:


```

local materialInterfaceGroupAttr = Interface.GetAttr("material.interface")

local targetName = "arnold"

for i = 0, materialInterfaceGroupAttr:getNumberOfChildren() - 1 do
    local parameterName = materialInterfaceGroupAttr:getChildName(i)
    Interface.SetAttr("material.meta." .. parameterName .. "." .. targetName,
        StringAttribute("parameters." .. parameterName))
end

```

Creating Shadows

Different renderers support different types of shadows. The most common types of shadows are:

- Raytraced
- Shadow Map
- Deep Shadow Map

As Katana supports any renderer that implements its Renderer API, the shadow types available depend on the renderer.

The renderers that ship with Katana support the following shadow types:

- Arnold: raytracing (default).
- PRMan: raytracing, shadow maps, deep shadow maps.

Raytracing Shadows in Arnold

As long as the light shader supports them, your Arnold renders use raytraced shadows by default.

You can turn off shadows by disabling them within the **Parameters > Linking** sub-tab of the Gaffer node.

Raytracing Shadows in PRMan

Raytracing shadows within PRMan involves two steps:

- set raytracing in the light,
- define which objects within the scene cast shadows.

Turning on Raytracing for PRMan Lights

To turn on raytracing:

Type raytrace in the material shader's shadow file parameter. For instance, the **KatanaSpotlight** shader uses a **lightShader > Shadows > Shadow_File** parameter.

Specifying Which Objects Cast Shadows in PRMan

To specify which objects cast shadows:

1. Create a PrmanObjectSettings node and connect it into the recipe.
2. Select the PrmanObjectSettings node and press **Alt+E**.
The PrmanObjectSettings node becomes editable within the **Parameters** tab.
3. Assign the scene graph locations of the shadow casting objects to the PrmanObjectSettings node's **CEL** parameter (see [Assigning Locations to a CEL Parameter](#)).
4. Select **Yes** in the **attributes > visibility > transmission** dropdown.



TIP: While raytracing shadows using the **KatanaSpotlight**, you can:

- change the light radius using **lightShader > Shadows > shadowblur**, or
- change the number of shadow rays using **lightShader > Shadows > shadownsamps**.

Creating a Shadow Map

A shadow map is a depth render from a light. This render is later used by the light shader to work out whether an object is occluded by another object by testing its depth from a light against the depth recorded in the shadow map.

To create a shadow map:

1. Create a ShadowBranch node and connect it into the recipe.
2. Create a RenderOutputDefine node and connect it below the ShadowBranch node.
3. Select the RenderOutputDefine node and press **Alt+E**.
The RenderOutputDefine node becomes editable within the **Parameters** tab.
4. Select file from the **locationType** dropdown.
5. Enter a file name for the shadow map in the **render Location** parameter.
When using the OpenColorIO standard, shadow map files should have an **_ncf** suffix. For more on OpenColorIO, see [Managing Color Within Katana](#).
6. Create a RenderSettings node and connect it below the RenderOutputDefine node.
7. Select the RenderSettings node and press **Alt+E**.
The RenderSettings node becomes editable within the **Parameters** tab.
8. Enter the scene graph location of the light whose shadow map you are creating in the **cameraName** parameter.
9. Select the shadow map resolution from the **resolution** dropdown or type in the resolution to the right of the dropdown.

10. Create a Render node and connect it below the RenderSettings node.
11. Right-click on the Render node and select **Disk Render** to render the shadow map to a file.

Creating a Deep Shadow Map

A deep shadow map is rendered from a light and stores the transparency levels as you step through the image until fully opaque. With the use of multiple samples, fine geometry and curves can cast realistic shadows (particularly useful for hair and fur). Motion blur can also be included within deep shadow maps.

A shadow map generates more information, and hence larger files, than a normal shadow map.

To create a deep shadow map:

1. Create a ShadowBranch node and connect it into the recipe.
2. Select **primary deepshad** from the **defineOutputs** dropdown.
3. Create a RenderOutputDefine node and connect it below the ShadowBranch node.
4. Select the RenderOutputDefine node and press **Alt+E**.
The RenderOutputDefine node becomes editable within the **Parameters** tab.
5. Select **file** from the **locationType** dropdown.
6. Enter a file name for the deep shadow map in the **renderLocation** parameter.
When using the OpenColorIO standard, shadow map files should have an **_ncf** suffix. For more on OpenColorIO, see [Managing Color Within Katana](#).
7. Create a RenderSettings node and connect it below the RenderOutputDefine node.
8. Select the RenderSettings node and press **Alt+E**.
The RenderSettings node becomes editable within the **Parameters** tab.
9. Enter the scene graph location of the light, whose deep shadow map you are creating, in the **cameraName** parameter.
10. Select the deep shadow map resolution from the **resolution** dropdown, or type in the resolution to the right of the dropdown.
11. Create a Render node and connect it below the RenderSettings node.
12. Right-click on the Render node and select **Disk Render** to render the deep shadow map to a file.

Using a Shadow Map in a Light Shader

A shadow map (whether deep or normal), once generated, is just a file. In order to utilize the file, a light shader must know where it is.

Connecting a Gaffer Light to a Shadow Map File

To connect a shadow map:

1. Select the Gaffer node and press **Alt+E**.
The Gaffer node becomes editable within the **Parameters** tab.
2. Select the light in the Gaffer's hierarchical view.
The lights parameters display below the hierarchical view.
3. Click the **Material** sub-tab in the light's parameters.
4. Click  next to the **material** parameter grouping.
The connected shaders list displays.
5. Click  next to the **lightShader** parameter.
6. Click  next to the **Shadows** parameter grouping.
7. **Shift**+middle-click and drag from the Render node that is generating the shadow file to the **Shadow_File** parameter.
An expression link is created between the output from the Render node and the **Shadow_File** parameter.



NOTE: **Shadow_File** is the parameter used for **KatanaSpotlight**, your shaders may use a different parameter name.



TIP: While using shadow maps with the **KatanaSpotlight**, you can:

- Blur the shadow map using **lightShader > Shadows > shadowblur**,
- Change the number of shadow map samples using **lightShader > Shadows > shadownsamps**, or
- Move the shadow map (to avoid artifacts) using **lightShader > Shadows > shadowbias**.

Positioning Lights

To position a light it first needs to be visible within the **Scene Graph** tab (see [Changing What is Shown in the Viewer](#)) then positioned within the **Viewer** tab.

Moving a Light Within the Viewer

To move a light, you can:

- Translate and rotate the light with the manipulators, (see [Transforming an Object in the Viewer](#)).

Or

- Look through the light and change its view position, (see [Changing what you Look Through](#)).

Getting to Grips with the Gaffer Node

The method described in [Creating a Light in Katana](#), although valid, would be slow for a large number of lights. Katana's Gaffer node wraps light creation into a single node and adds the ability to:

- Create more than one light.
- Use light profiles for different types of light.
- Add rigs to group lights together.
- Mute and solo lights and rigs.
- Link lights to specific objects.
- Add aim constraints to lights.



NOTE: Some of the options listed in [Creating a Light Using the Gaffer Node](#) may not be available due to the extensive customizability of Katana. Some of the Gaffer node's menu options are created using profiles which can result in different light creation menu options.



NOTE: Using both Gaffer and GafferThree nodes together in a single node graph is not supported as it can result in unexpected mute and solo behavior.

Creating a Light Using the Gaffer Node

To create a light with the Gaffer node, you need first to create a light and then add a light shader to that light:

1. Create a Gaffer node and place it within the project.
2. Select the Gaffer node and press **Alt+E**.
The Gaffer node becomes editable within the **Parameters** tab.
3. Right-click the **gaffer** location in the Gaffer node's parameter hierarchy and select **Add > Add Light**.
4. Click **<none>** below the **Shader** heading in the parameter hierarchy and select **KatanaSpotlight** from the list.



NOTE: **KatanaSpotlight** is a simple PRMan light shader that ships with Katana. Depending on your studio's setup, you may need to choose another light shader.

Making Use of Rigs

Rigs create a scene graph group complete with transform attributes and the ability to easily add constraints. Lights created below the rig inherit its transformations which enables you to move the lights around as one. Rigs can also be exported and imported.

Creating a Rig

Right-click the **gaffer** location in the Gaffer node's parameter hierarchy and select **Add > Add Rig**.

A rig is created below the **gaffer** in the parameter location. It is possible to nest rigs by right-clicking a rig location instead of the **gaffer** location.

Adding a Light to a Rig

In the Gaffer object table, right-click on a rig and select **Add > Add Light**.

A light is created under the rig location.



NOTE: You can expand or collapse the lights nested under a rig location. Simply right-click on a rig and select the required option from the dropdown menu.

Importing a Rig

1. Right-click a location within the Gaffer node's parameter hierarchy, such as **gaffer**, and select **Add > Import Rig...**

The **Import Rig** dialog displays.

2. Select the rig file in the dialog and click **Import**.

The rig is imported below the selected location.

Exporting a Rig

1. Right-click on the rig to export and select **Export Rig...**

The **Save Rig** dialog displays.

2. Navigate within the dialog to where you wish to save the light rig and enter a rig name.

3. Click **Save**.

Rigs are saved with a **.gprig** file extension.

Adding a Point Constraint to a Rig

1. Select the rig and click the **Parameters > Object** sub-tab.
2. Check **enable point constraint** and open the **point constraint options** parameter grouping.
3. Enter the scene graph location for the target in the **targetPath** parameter. For more on using scene graph location parameters, see [Manipulating a Scene Graph Location Parameter](#).
4. Select from the **targetOrigin** dropdown the part of the target to use as the point constraint:
 - **Object** - the object's transform position is used.

- **Bounding Box** - the center of the object's bounding box is used.
- **Face Center Average** - the center of all the faces for the object are averaged to create the point constraint position.
- **Face Bounding Box** - the center of the face's bounding box is used.

Adding an Orient Constraint to a Rig

To add an orient constraint:

1. Select the rig and click the **Parameters** > **Object** sub-tab.
2. Check **enable orient constraint** and open the **orient constraint options** parameter grouping.
3. Enter the scene graph location for the target in the **targetPath** parameter. For more on using scene graph location parameters, see [Manipulating a Scene Graph Location Parameter](#).
4. Select the axes to constrain (by default, it's all three). To remove the constraint for:
 - the x-axis, select **No** from the **xAxis** dropdown.
 - the y-axis, select **No** from the **yAxis** dropdown.
 - the z-axis, select **No** from the **zAxis** dropdown.

Defining a Master Light Material

At times it is best to have a master material and set local overrides per light. This can be done within the Gaffer node by creating a master material and assigning it to a light. Any changes made within the light's **Material** sub-tab act as an override for the master.

Creating a Master Material

Inside the Gaffer node's hierarchy, right-click and select **Add** > **Add Master Material**.

A master material location is created inside the Gaffer node. Its shaders and attributes are only visible within the scene graph when it is assigned to a light.

Assigning a Master Material to a Light

Click **<none>** below the **Shader** heading in the parameter hierarchy, in line with the light, and select the master material from the list (master materials are displayed in green).

Adding a Sky Dome Light

Sky domes are generally used for image based lighting where an image is placed around the scene and points from the sky dome provide illumination. Their exact use depends on the shader assigned.

To add a sky dome, right-click and select **Add > Add Sky Dome** inside the Gaffer node's hierarchy.

A sky dome is added to the Gaffer node's hierarchy.



NOTE: A sky dome needs a suitable shader assigned. The shaders available depend on your studio.

Adding an Aim Constraint to a Light

Lights created inside the Gaffer node come with the ability to use an aim constraint. Using an aim constraint makes the light point at an object (the target) within the scene.

Enabling an Aim Constraint for a Gaffer Light

1. Select the light within the **gaffer** hierarchy inside the **Parameters** tab.
2. Select the **enable aim constraint** checkbox within the **Object** sub-tab.
The **aim constraint options** parameter grouping displays.
3. In the **aim constraint options** parameter grouping, enter the aim target in **targetPath** (for more information on how to enter a scene graph location, see [Manipulating a Scene Graph Location Parameter](#)).

Changing the Aim Constraint's Center Point

Select from the **targetOrigin** dropdown:

- **Object** - the point defined by the transform of the object.
- **Bounding Box** - the center of the bounding box.
- **Face Center Average** - the average from all the face centers.
- **Face Bounding Box** - the bounding box of all the faces.



NOTE: Using **Face Center Average** or **Face Bounding Box** could be slow for heavy geometry.

Creating an Aim Target

Inside the Gaffer node's hierarchy, right-click and select **Add > Add Aim Target**.

A locator is created that can be used as the target for an aim constraint.

Linking Lights to Specific Objects

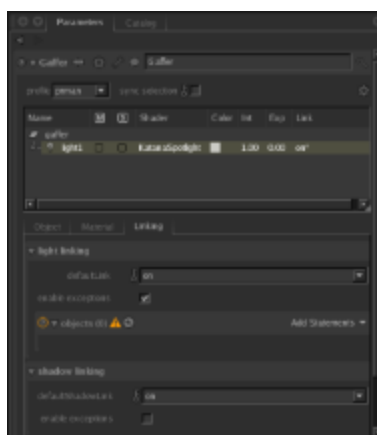
Light linking allows you to light a set of objects while others aren't or turn off the lights on a set of objects while the others are lit.

Setting the Default Behavior for a Light

1. Select the light within the **gaffer** hierarchy inside the **Parameters** tab.
2. Inside the **Linking** sub-tab, select **light linking > defaultLink** :
 - **on** - everything is lit by default; exceptions are unlit.
 - **off** - everything is unlit by default; exceptions are lit.

Creating a Light's Exception List

1. Toggle the **enable exceptions** checkbox to **on**.
2. Assign the scene graph locations of the objects to be excluded to the **light linking > objects** parameter (see [Assigning Locations to a CEL Parameter](#)).



Linking Shadows to Specific Objects

Linking shadows is handled in the same manner as linking lights above. Each location within the scene graph below **/root/world** has a **lightList** attribute. This is where light linking and shadow linking information is stored.



NOTE: Currently only the Arnold renderer supports shadow linking within Katana.

Deleting from the Gaffer Object Table

To delete an item from the Gaffer hierarchy, right-click on the item in the hierarchy and select **Delete** or press the **Delete** keyboard shortcut.

Locking a Light or Rig's Transform

Once a light is in the correct position, you can lock it to prevent accidental movement. Locking a light does not prevent it from being edited or deleted.

To toggle whether a light is locked, right-click on the light and select **Lock**.

The blue lock  icon is either added or removed in place of the light icon in the Gaffer object table.



NOTE: This also works for light rigs and aim targets.

Soloing and Muting Lights and Rigs

You can either solo or mute lights in a scene. Soloing a light means that you are only keeping that one specific light to light the scene. All the other lights are automatically set as "mute" and therefore turned off. Muting lights means you can turn specific lights off while others are lit. Soloing or muting rigs works the same way.

Soloing Lights and Rigs

To solo a light, do the following:

1. In the Gaffer object table, select the light you want to solo.
2. In the Solo column  check the box in-line with the selected light.

The light is set as "solo" and all the other lights are automatically set as "mute".

In the Mute column , the icon  shows which lights are set as "mute" and it is also displayed in the **Scene Graph** tab to indicate which light locations are in a mute state.



NOTE: Soloing lights with the Gaffer is non-persistent across sessions.



NOTE: To solo a rig, follow the same procedure but select a rig instead of a light.



NOTE: Soloing a light in a GafferThree node affects any lights created in other downstream GafferThree nodes.

Muting Lights and Rigs

To mute a light, do the following:

1. In the Gaffer object table, select one or as many lights as you want to mute.
2. In the Mute column  check the box in-line with the selected light or one of the selected lights for multiple selection.

The selected lights are set as "mute" while all the other lights still light the scene.

In the Mute column , the icon  shows which lights are set as "mute" and it is also displayed in the **Scene Graph** tab to indicate which light locations are in a mute state.



NOTE: To mute one or more rigs, follow the same procedure but select rigs instead of lights.

Duplicating an Item Within the Gaffer Object Table

To duplicate an item within the Gaffer hierarchy, right-click on the item in the hierarchy and select **Duplicate**.

The item is duplicated into the Gaffer object table and a number is appended to the name.

Syncing the Gaffer Selection with the Scene Graph

When editing lights using both the Gaffer and manipulators in the Viewer, it is often convenient to sync the selection of those lights between the two locations. The sync selection parameter in the Gaffer node's parameters allows you to set sync selection in three ways:

- **off** - no syncing is performed (the default).
- **out** - selection of a light in the Gaffer node is mirrored in the **Scene Graph** tab, but not the other way around.
- **in/out** - selecting in either the **Scene Graph** tab or Gaffer node results in the corresponding entry in the other also being selected.

Rendering Your Scene

Overview

Katana was developed from the ground up to address the problems of scalability and flexibility. Solving both problems was essential for dealing with the demands of highly complicated modern productions.

When it comes to rendering, Katana's renderer-agnostic nature provides the flexibility to allow CG supervisors and pipeline engineers to select the appropriate renderer for the show or shot. The renderer connects to Katana through a renderer-specific plug-in. Currently, Katana ships with renderer plug-ins for both PRMan and Arnold and an API that allows developers to support other renderers or versions of PRMan or Arnold not currently supported.

Scalability is also at the heart of rendering in Katana. PRMan and Arnold support procedurals that can be evaluated on demand (often called deferred evaluation) and are able to recursively call other procedurals. At render time, they are passed scene descriptions in the form of a procedural recipe to be run inside the renderer. Through this approach, very large scenes are easier to manage, and the resources needed to deal with them are reduced. In addition, deferred evaluation significantly simplifies pipelines by removing the need to write large scene data files (such as RenderMan **.rib** files and Arnold **.ass** files) for each frame before rendering starts. Renderers that don't support these features are still usable with Katana, but they don't leverage its full benefit.

Render Types

Katana has a number of context sensitive render options, available through the right-click menu on nodes. Renderer plug-ins advertize the methods they support, so the right-click menu render options shown depend on the node selected and the methods advertized in the renderer plug-in. All render options send the scene graph, as generated at the selected node, to the selected production renderer. The exact options you see may vary, depending on the configuration of your studio's plug-ins, but the default set is:

Preview Rendering

- **Preview Render**

The render is a static image, displayed in the **Monitor** tab. The image is not written to disk.

Live Rendering

• Live Render

The difference between a Preview Render, and a Live Render is that under Live Render, changes to the camera, lights, materials, or geometry transformations result in updates to the image displayed in the **Monitor** tab. See [Changing How to Trigger a Live Render](#) for more on which activities trigger a Live Render, and how to edit them.



NOTE: PRMan does not currently support geometry transformation updates when Live Rendering.



NOTE: Motion blur in Live Rendering is not supported for interactive cameras. To enable motion blur in a live render session, set the camera's **makeInteractive** parameter to **No**.
If you bring in an animated camera through an Alembic or other external file, the camera keeps its animation, even when **makeInteractive** is set to **No**.



TIP: To stop any render - including Live Rendering - either press the **Esc** key, or select **Render > Cancel Renders** in the menu bar.

Alternatively, starting a new Live Render automatically stops the previous Live Render and doesn't need to be specifically canceled using either of the methods above.

Disk Rendering

• Disk Render

The scene is written to disk, at the location specified in a Render node, and for this reason, is only available from Render nodes.

• Disk Render with Dependencies

Writes a Disk Render, along with any dependencies of the Render node, to disk.

• Disk Render Dependencies Only

Renders just dependencies to disk.

Disk Render Dependencies Before

Katana offers the option of rendering any dependencies before either Preview or Live Rendering. See [Setting up Render Dependencies](#) for more on dependencies.

3D nodes have a right-click menu sub-heading, **Disk Render Dependencies Before** which holds the following options:

• Preview Renders

When selected, render dependencies (such as shadow maps) are rendered to disk before performing a Preview Render.

- **Live Renders**

When selected, render dependencies are rendered to disk before Live Rendering.

Disk Render Upstream Render Outputs

Nodes that have rendered 2D images from other Katana nodes as dependencies have a right-click menu sub-heading, **Disk Render Upstream Render Outputs** which holds the following options:

- **Preview Renders: Unless Already Cached**

When selected, during a Preview render all incoming image dependencies are rendered to disk, unless they have already been rendered to disk and cached.

- **Preview Renders: Always**

When selected, during a Preview render all incoming image dependencies are rendered to disk, regardless of whether they are already cached or not.

- **Disk Renders: Always**

This is for information only. This option cannot be changed.

During a disk render, all incoming image dependencies are rendered to disk, regardless of whether they are already cached or not.

Render Farm

Certain nodes, such as Render and ImageWrite, have a right-click menu sub-heading **Render Farm**, which - by default - holds the following options:

- **Generate Farm XML File for Current Node...**
- **Generate Farm XML File for Selected Nodes...**
- **Generate Farm XML File for All Nodes...**



NOTE: The options available under right-click > **Render Farm** depend on the FarmAPI. The options and function you see may vary.

Debugging

3D nodes have a right-click menu subheading **Debugging**, which offers options to view debug information in a text editor. The options are:

- **Debugging > Open Filter Text Output in** <your text editor>
Displays the Katana filters used to traverse the scene graph.
- **Debugging > Open** <your renderer's debug file type> **Output in** <your text editor>

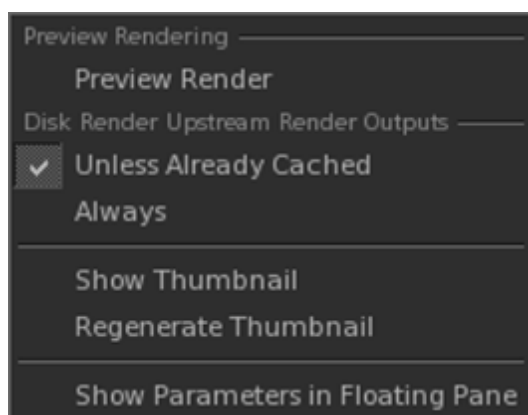
Displays the debug file type of your selected renderer.

Render Type Availability

2D Image Nodes

For 2D image nodes (such as ImageGamma or ImageCrop) the right-click menu render options are:

- **Preview Rendering**
 - Preview Render
- **Disk Render Upstream Render Outputs**
 - Unless Already Cached
 - Always

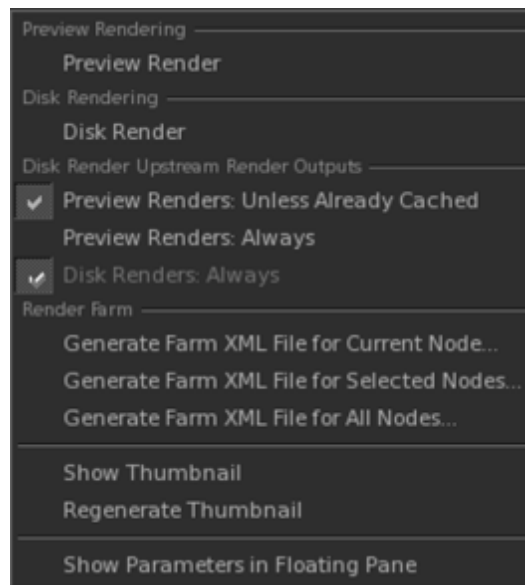


2D Image Write Nodes

The render types available from an ImageWrite node are:

- **Preview Rendering**
 - Preview Render
- **Disk Rendering**
 - Disk Render
- **Disk Render Upstream Render Outputs**
 - Preview Renders: Unless Already Cached
 - Preview Renders: Always
 - Disk Renders: Always
- **Render Farm**
 - Generate Farm XML File for the Current Node...
 - Generate Farm XML File for Selected Nodes...

- Generate Farm XML File for All Nodes...



3D Nodes

The render types available from 3D nodes (such as a PrimitiveCreate, CameraCreate, or Material nodes) are:

- **Preview Rendering**

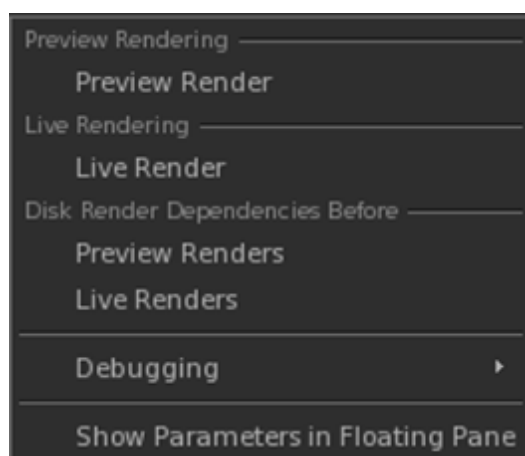
- Preview Render

- **Live Rendering**

- Live Render

- **Disk Render Dependencies Before**

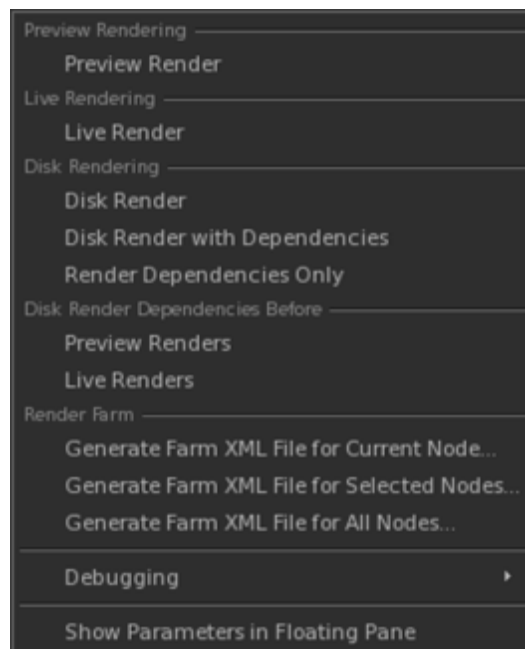
- Preview Renders
- Live Renders



Render Node

The render types available from Render nodes are:

- **Preview Rendering**
 - Preview Render
- **Live Rendering**
 - Live Render
- **Disk Rendering**
 - Disk Render
 - Disk Render Dependencies
 - Render Dependencies Only
- **Disk Render Dependencies Before**
 - Preview Renders
 - Live Renders
- **Render Farm**
 - Generate Farm XML File for Current Node...
 - Generate Farm XML File for Selected Nodes...
 - Generate Farm XML File for All Nodes...



Influencing a Render

Some key nodes for influencing a render are:

- **RenderOutputDefine** - used for setting the primary output, or adding secondary render outputs (such as color, point cloud, shadow), the channel (including arbitrary output variables - AOVs), the colorspace, the pass name, and the final render destination. The node changes attributes under **renderSettings.outputs** at the **/root** location. See [Setting up a Render Pass](#) for more on this.
- **RenderSettings** - sets the render camera, the choice of renderer, and the resolution. This node changes the **renderSettings** attribute at the **/root** location.
- **<YourRenderer>GlobalSettings** - sets renderer specific globally scoped settings. For instance, use the **ArnoldGlobalSettings** node when rendering with Arnold. This node's parameters sets attributes stored under the **<xxx>GlobalStatements** attribute at **/root**, for instance **arnoldGlobalStatements**.
- **<YourRenderer>ObjectSettings** - sets renderer specific options which apply to selected locations within the scene graph. For instance, the **PrmanObjectSettings** node is used when rendering with PRMan.

Performing a Render

You can trigger a render from the Katana UI by choosing one of the right-click contextual render options (bearing in mind that not all options are available from all nodes).

You can cancel any render by pressing the **Esc** shortcut, or choosing **Render > Cancel Renders**.

You can repeat the previous render by pressing the **\ (backslash)** shortcut, or choosing **Render > Repeat Previous Render**.

Executing Renders with the Render Node

The Render node acts as a render point within a recipe.

To write a render pass to disk:

1. Create a Render node and add it to the recipe.
Add the Render node at the point in the recipe where you are happy with the interactive render.
2. Optionally, add a RenderOutputDefine above the Render node to define output name, format, and file location.
3. Right-click on the Render node and select **Disk Render** or **Disk Render with Dependencies**.

The scene graph is generated up to that node and sent to the renderer. The render is saved to your temp directory or, if your recipe has a RenderOutputDefine node upstream of the Render node, the rendered output is saved to the locations specified there.



NOTE: Unlike a Preview Render or Live Render (which show renders in the **Monitor** tab as they're generated), the results of a Disk Render are only visible after the render is complete.

Rendering From the Command Line

To render a scene from the command line, you can use both Katana's **Batch** and **Script** modes:

- **Batch** mode - You must provide a filename or asset ID specifying the Katana project to render, the name of the Render node in the specified Katana project from which to render, and the frame range to render. For example, the following command renders the specified Katana project from the MyRenderNode node for frames **1** to **10**:

```
katana --batch --katana-file=/path/to/myscene.katana
--render-node=MyRenderNode -t 1-10
```

- **Script** mode - This mode allows you to execute a Python script in Katana's Python environment, so you can perform more complex actions such as changing parameters, creating nodes, or modifying node connections, as well as launching renders.



NOTE: For more information on how to use **Batch** and **Script** modes, see the *Katana Launch Modes* chapter in the *Katana Technical Guide*.

Performing a Preview Render

You can perform a Preview Render at any 3D node within the recipe. A scene description is generated up to that node (to what extent the scene is generated or deferred depends upon the renderer). The scene description is then sent to the actual production renderer, and the results are visible in the **Monitor** tab (see [Viewing Your Renders](#)). To perform a preview render:

1. Right-click on a 3D node within .
2. Select **Preview Render**.

You can also trigger a Preview Render from the currently viewed node by pressing the **P** shortcut, or choosing **Render > Render View Node**.

Performing a Live Render

Live rendering is useful for getting immediate feedback on changes you make to cameras, lights, and materials. Within a Live Render session, changes to materials and object transformations on specified scene graph locations are communicated to the renderer.

Starting and Stopping Live Rendering

To start live rendering:

1. Right-click on any node.
2. Select **Live Render**.
3. The image in the **Monitor** tab is updated in response to your actions in Katana according to the selected **3D Update Mode**.

To stop live rendering:

1. Press the **Esc** key
OR
2. Select **Render > Cancel Renders**.
OR
3. Click the stop button  next to the progress bar of the render thumbnail in the **Monitor** tab.



NOTE: Triggering a new Live Render while a previous Live Render is running, automatically cancels the previous render in favor of the new one.

Controlling Live Rendering

You can control Live Rendering behavior in a number of ways using several options in the **Scene Graph**, **Monitor**, and **Viewer** tabs as well as in the menu bar. For example, you can change which material and light edits trigger a Live Render, and when Live Render updates should take place. To start the Live Render, right-click on any node and select **Live Render**.

Live Rendering options can be found in the following places:

- In the **Scene Graph** tab, you can select which location generates Live Rendering updates when their attributes change.
- In the **Monitor** and **Scene Graph** tabs as well as in the menu bar, you can choose how Live Rendering should take place with the **3D Update Mode**.
- In the **Viewer** tab, you can change from which render view point to Live Render.



NOTE: Not all nodes have an immediate effect on the Live Render. For example adding a PrimitiveCreate node does not cause the new primitive to appear because adding new geometry is not supported in the render plug-ins.



NOTE: The view node changes in the **Node Graph** tab are not reflected in the **Scene Graph** tab when the **3D Update Mode** is set to **Manual**.

Global Options

Selecting the Live Render Camera

You can change the render view point at any stage in the Live Render process using the **Look Through Lights and Cameras** menu in the **Viewer** tab.

To activate the **Live Render from viewer camera** option, do the following:

1. In the **Viewer** tab, click on the toggle button .

The toggle button is now blue  indicating that the **Live Render from viewer camera** option is activated.

2. In the **Look Through Lights and Cameras menu**, select the camera or light you want the Live Render to render from.

The image in the **Monitor** tab is updated in response to your actions in the **Viewer** tab.

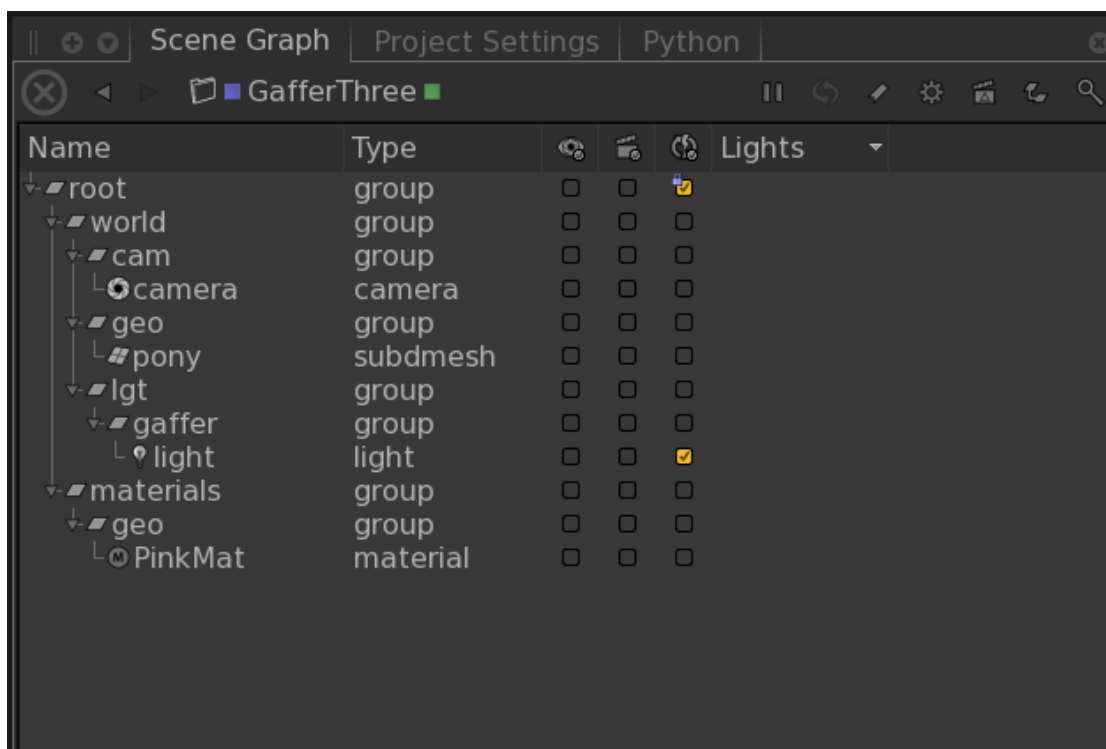


NOTE: The **Live Render from viewer camera** option supports **Continuous** and **Pen-up** modes at the moment but not **Manual** mode.

Selecting Which Lights Trigger Updates

To specify which light changes you want to send to the renderer, do the following:

In the **Scene Graph** tab, tick the relevant boxes in the Live Render Updates  column depending on which light changes you want to send to the renderer.

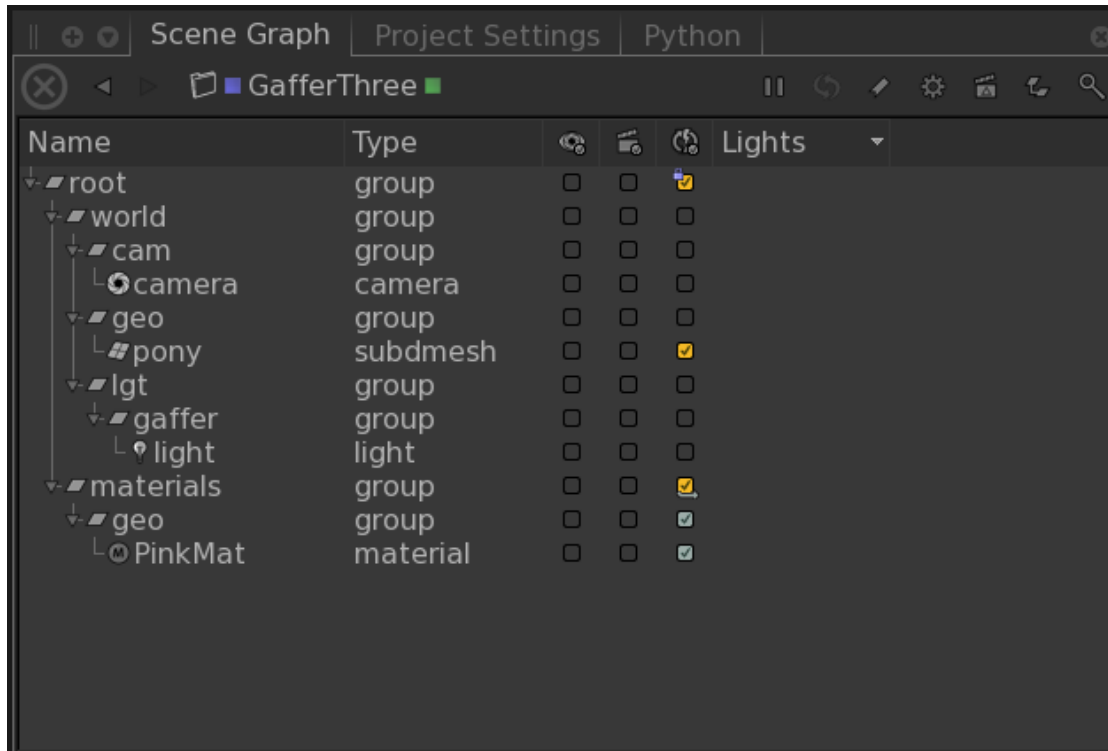


NOTE: If a light is to be used as the Live Render camera, it is enabled in the **Scene Graph Live** column.

Selecting Which Materials Trigger Updates

To specify which material changes you want to send to the renderer, do the following:

In the **Scene Graph** tab, tick the relevant boxes in the Live Render Updates  column depending on which material changes you want to send to the renderer.



NOTE: In the **Scene Graph** tab, when you want to send changes of a material to the renderer you must also select the geometry location to which the material is applied for the Live Render to process it.

Changing How to Trigger a Live Render

There are three different ways of triggering a Live Render update. You can select your preferred method from the **3D Update Mode** dropdown in the **Scene Graph**, **Monitor**, and **Viewer** tabs as well as in the menu bar. The options are:

- **Manual**

Changes to materials, lights, or geometry transformations do not trigger an update of the scene. To have the changes take effect, click the **Trigger 3D Update** button .

- **Pen-Up**

Changes to materials, lights, or geometry transformations trigger an update of the scene only when the mouse button is released or a parameter change is applied.

- **Continuous**

Changes to materials, lights, or geometry transformations, including some manipulations in the **Viewer** tab, continuously trigger an update of the scene.

3D Update Mode on	Indicators
Manual mode	
Pen-up mode	
Continuous mode	



NOTE: Changes to the Node Graph topology are not supported by the shipped renderer plug-ins, and are not reflected in any Live Render updates. PRMan does not support geometry transformations during Live Rendering.



NOTE: As the **Manual 3D Update Mode** currently defers all scene graph cooking in response to 3D parameter edits, parameter interfaces that rely on scene graph data, such as GafferThree's scene graph view and shader selection interfaces, don't update whilst certain parameter edits are pending. This mode is therefore only suggested for use while editing individual parameters in the **Parameters** tab or while manipulating objects in the **Viewer** tab.

Taking a Snapshot of the Current Render

You can create an entry in the **Catalog** tab for the current render image. To add the entry to the **Catalog** tab, click the **Create Snapshot in Catalog** button in the **Live Render** menu in the **Monitor** tab.

Renderer Specific Options

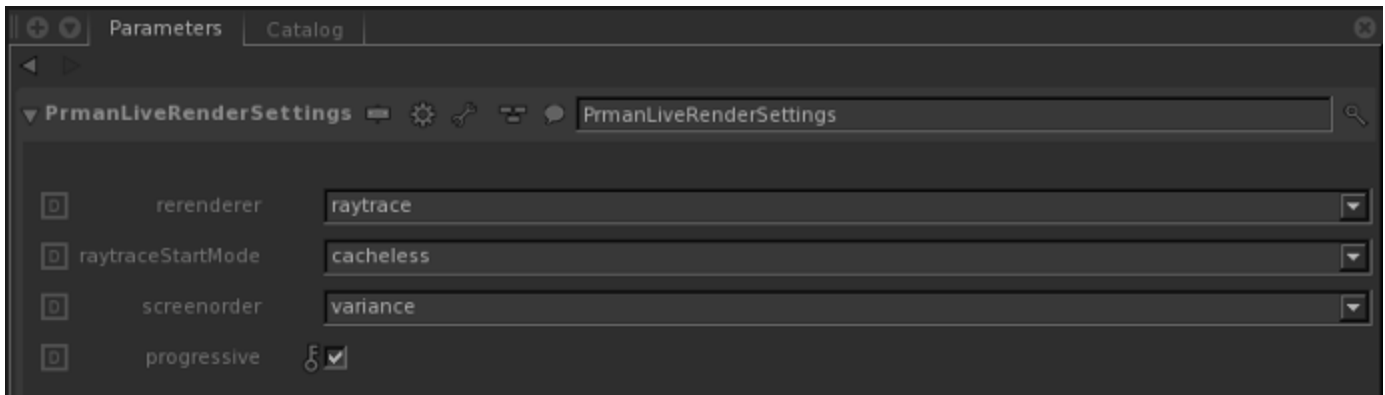
Although many of the options in the Live Render process are the same between renderers, some renderer-specific options allow features of the Live Render to be exploited. Also, renderers support a different set of Live Render features, and in some cases they implement the same features differently.

Additional live rendering options are exposed under renderer-specific nodes called <renderer>LiveRenderSettings. You can also find other renderer-specific settings in the **Monitor** tab, in the **Live Render** menu, under your renderer entry.

PRMan-Specific Options

To use PRMan Live Rendering options, do the following:

Add a PrmanLiveRenderSettings node to the project and connect it to the **Node Graph** tree. Select the node and press **Alt + E** to make it editable in the **Parameters** tab. In the **Parameters** tab, you can see the following PRMan options:



- **rerenderer**

Sets whether PRMan uses the **Raytrace** or **Reyes (Scanline)** render filter. The default is **Raytrace**.

- **raytraceStartMode**

Sets whether the renderer uses a cache. The default is **Cacheless**. When set to **From Cache**, you can change the Live Render cache location by selecting the PrmanGlobalSettings node's **options > render > rerenderbake** checkbox and typing the directory in the resulting **options > render > rerenderbakedbdir** parameter and optionally add a cache namespace in **options > render > rerenderbakedbname**. If not specified, the default temp directory is used, but the cache does not persist after Katana is closed.

- **screenorder**

Sets in what order the bucket updates are received in the **Monitor** tab. The options from the dropdown are:

- **Variance (Prioritize Edges and Strong Contrasts)**
- **Bucket Order**

The default is **Variance**.

- **progressive**

Enables or disables the PRMan Progressive Refinement parameter. When enabled, multi-resolution images produced by re-rendering are displayed. When disabled, all rendering is at the highest resolution. The default is **Enabled**.

Arnold-Specific Options

To use Arnold Live Rendering options, do the following:

Click on the **Live Render** menu in the **Monitor** tab, then click on **Arnold**. The **Arnold** entry contains the following options:

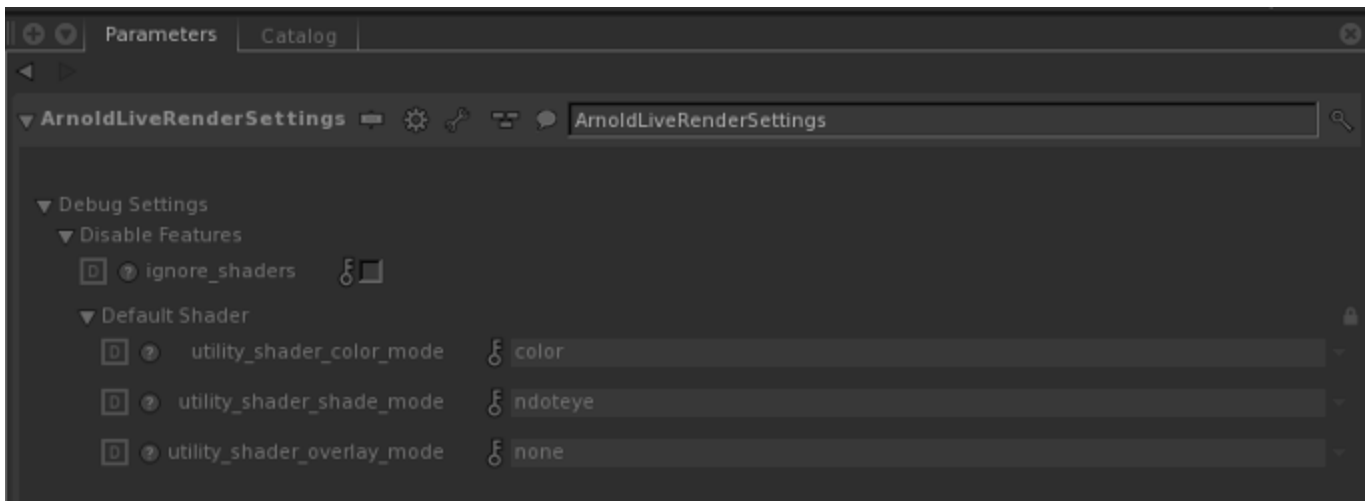
- **Flush Texture Cache**

Flushes the texture cache and reloads any textures.

- **Pause Rendering Until Update**

Stops the current render and waits for another light, material, geometry transform, or camera edit to trigger a new Live Render.

Add an `ArnoldLiveRenderSettings` node to the project and connect it to the **Node Graph** tree. Select the node and press **Alt + E** to make it editable in the **Parameters** tab. In the **Parameters** tab, you can see the following Arnold options:



- **ignore_shaders**
Sets whether to override the current utility shader color mode and shade mode.
- **utility_shader_color_mode**
Sets the scene data to visualize for debugging.
- **utility_shader_shade_mode**
Sets the base shading mode for visualization.
- **utility_shader_overlay_mode**
Combines an additional color mode on top of the base mode.



NOTE: For further information on Live Rendering with a particular renderer, see its accompanying documentation.

Setting up Interactive Render Filters

Interactive render filters enable you to set up common recipe changes for interactive render and live renders without having to add them at each point in the recipe you want to test. These filters are ignored for disk renders.

For example, you can set up an interactive render filter to reduce the render image size, thus making debugging and light tests much quicker. Other examples might include anti-aliasing settings, shading rate changes (if using

These filters are bundled together inside the InteractiveRenderFilters node and toggled using at the top of the interface.

1. Create an InteractiveRenderFilters node and place it anywhere within the **Node Graph**.

- The InteractiveRenderFilters node becomes editable within the **Parameters** tab.

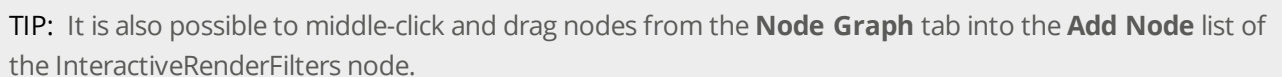
- A new **RenderFilter** is created.

5. To create a group of filters, type a group name in the **category** parameter.

- Filter the list using the **Categories** dropdown or the **Filter** field.

-
- The screenshot shows the Unity Inspector window with the 'InteractiveRenderableBars' object selected. The 'Render Filter' section on the left shows the 'RenderSettings' component. The 'Render Settings' section on the right shows the 'Renderable' component with the following settings:
- name:** `Renderable`
 - category:** `Renderable`
 - RenderableSettings:**
 - cameraMask:** `0`
 - renderOrder:** `0`
 - resolution:** `256` (with a dropdown menu showing '256x256')
 - depthTexture:** `0`
 - adjustForScreenResolution:** `No adjustment`
 - maxTextureSamples:** `1`
 - textureTypes:** `0`
 - textureSize:** `0`
 - compression:** `0` (with a dropdown menu showing '0')
 - renderQueue:** `0` (with a dropdown menu showing '0')

9. Repeat steps 3 to 7 for any additional filters.



Activating and Deactivating a Render Filter

By default, render filter nodes aren't active. To toggle whether a render filter is active:

1. Click  at the top of the interface.
2. Middle-click and drag filters from one side to the other to toggle whether they are active. (You can remove all the active filters by clicking the **clear** button.)

Setting up a Render Pass

By default, Katana starts with a primary render output (sometimes called the default pass).

The `RenderOutputDefine` node is used to define render outputs inside Katana. With it, you can set:

- The type of render output (such as color or point cloud (ptc)).
- The output's file type, colorspace, and location.
- The output's name.

All the attributes for a render pass are stored at the `/root` location under the **`renderSettings.outputs`** attribute. For instance, the primary pass attributes are stored under **`renderSettings.outputs.primary`**.

Defining and Overriding a Color Output

The `RenderOutputDefine` node can be used to create a new render output or override the settings for an existing one.

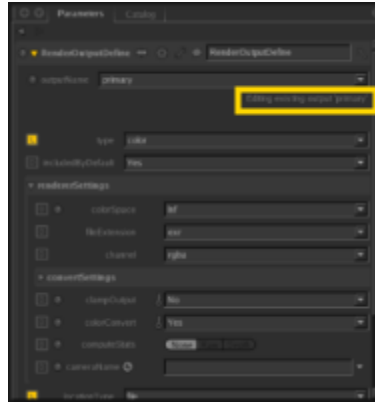
To define or override a color output:

1. Create a `RenderOutputDefine` node and add it to the recipe.
2. Select the `RenderOutputDefine` node and press **Alt+E**.

The `RenderOutputDefine` node becomes editable within the **Parameters** tab.

3. Type the pass name to define or override in the **outputName** parameter.

The **primary** pass is the default pass. Setting the pass name to something other than **primary** results in more than one pass. Katana provides feedback below the **outputName** parameter that displays whether or not you are creating a new pass or editing an existing one.



4. Select the output file's colorspace using the **colorSpace** dropdown.

The output colorspace is ignored if the **colorConvert** dropdown is set to **No**. For more on colorspace within Katana, see [Managing Color Within Katana](#).

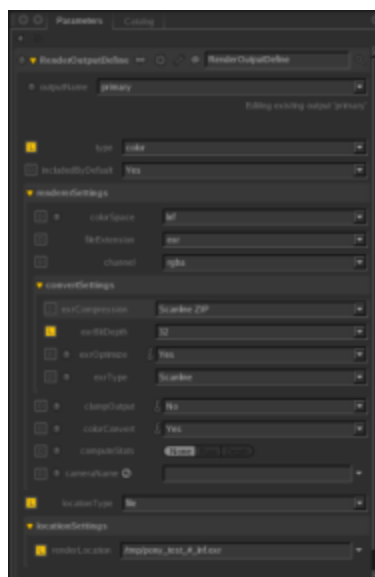
5. Select the file type to use from the **fileExtension** dropdown.

The file type should have sufficient bit-depth for the colorspace selected in step 4. For instance, the **Inf** colorspace requires 32-bits and, as such, some file formats aren't supported. Use the **convertSettings** parameter grouping to access the file type specific settings, including bit depth.

6. Select the type of location for the output file using the **locationType** dropdown.

The **locationType** can be:

- **local** - the output is saved to a temporary directory below **/tmp**. The exact directory is stored in the **KATANA_TMPDIR** environment variable.
- **file** - the **locationSettings** parameter grouping gains a **renderLocation** parameter where a file location can be specified.
- **studio's asset manager** - your studio may have an asset manager which is displayed here, details are implementation specific.



Defining Outputs Other than Color

The exact options available in the RenderOutputDefine node's **type** parameter depends on the current renderer. Each renderer plug-in is queried for the list of output types it supports.

The types available for PRMan are:

- **color**
Used for most renders.
- **deep**
Used for deep shadow map creation, see [Creating a Deep Shadow Map](#).
- **shadow**
Used for normal shadow map creation, see [Creating a Shadow Map](#).
- **raw**
Used when no color management is needed and allows you to directly set the **Display** line as it is output into the PRMan RIB stream.
- **ptc**
Used to define a point cloud output (although the actual point cloud is created by a shader).
- **script**
Used to inject a command line script into the render process that depends on a previous render (usually for txmake, ptfilter, or brickmake commands).
- **prescript**
Used to inject a command line script into the render process that runs before the render is started.
- **merge**
Used to merge a number of render outputs (usually AOVs) into a single OpenEXR file.
- **none**
Clears the pass, removing it from the output list.

The types available for Arnold are: **color**, **raw**, **script**, **prescript**, **merge**, and **none**. They behave in the same manner as their PRMan counterparts.

Defining an AOV Output

Arbitrary output variables (AOVs) allow data from a shader or renderer to be output during render calculations to provide additional options during compositing. This is usually data that is being calculated as part of the beauty pass, so comes with no extra processing cost. The ability to define AOVs is fully supported in Katana and is easy to set up.

To define an AOV output:

1. Follow [Defining an AOV Output](#) to set up a normal output.

2. In the **channel** parameter of the RenderOutputDefine node, enter the name of the AOV (this is the actual variable name that is output from the renderer), such as **_occlusion** or **P**.
3. Create a renderer specific OutputChannelDefine node, for instance PrmanOutputChannelDefine, and add it to the recipe above the RenderOutputDefine node.
4. Select the <Renderer>OutputChannelDefine node and press **Alt+E**.
The <Renderer>OutputChannelDefine node becomes editable within the **Parameters** tab.
5. Enter the same name as the **channel** parameter in step 2 into the **name** parameter (such as **_occlusion** or **P**).
6. At this point, the parameters needed by the renderer-specific OutputChannelDefine node vary depending on the renderer:
 - For an PrmanOutputChannelDefine node:
Select the data type of the AOV from the **type** dropdown.
 - For an ArnoldOutputChannelDefine node:
Make sure the parameters match the data type of the AOV. Consult the *Katana Reference Guide* that accompanies this User Guide for details on the various parameters.

Previewing Interactive Renders for Outputs Other than Primary

By default, the output displayed in the **Monitor** tab after an **Interactive Render** is the output from the **primary** pass. When additional outputs are available, such as from AOVs, you can view those in the **Monitor** tab alongside the primary pass.

To view additional interactive render outputs:

1. If there isn't a RenderSettings node below the RenderOutputDefine node then create one and add it.
2. Select the RenderSettings node and press **Alt+E**.
The RenderSettings node becomes editable within the **Parameters** tab.
3. Select the outputs to view from the **interactiveOutputs** parameter's list.
All outputs selected are available in the **Monitor** tab the next time you perform an interactive render downstream of this RenderSettings node. For more on viewing these renders, see [Selecting Which Output Pass to View](#).

OpenEXR Header Metadata

You can add arbitrary metadata to OpenEXR headers. The metadata must be set at attribute level - rather than through the UI - by creating attributes under **exrheaders**. For example, use an AttributeScript node targeting the **/root** location to set the following:

```
SetAttr( "renderSettings.ouputs.primary.rendererSettings.exrheaders.test_string", [
  "Your string" ] )

SetAttr( "renderSettings.ouputs.primary.rendererSettings.exrheaders.test_int", [ 1 ] )
```

```

SetAttr( "renderSettings.ouputs.primary.rendererSettings.exrheaders.test_int4", [ 1, 2,
3, 4 ] )
SetAttr( "renderSettings.ouputs.primary.rendererSettings.exrheaders.test_float", [ 5.7
] )
SetAttr( "renderSettings.ouputs.primary.rendererSettings.exrheaders.test_float2", [
3.8, 2.5 ] )

```



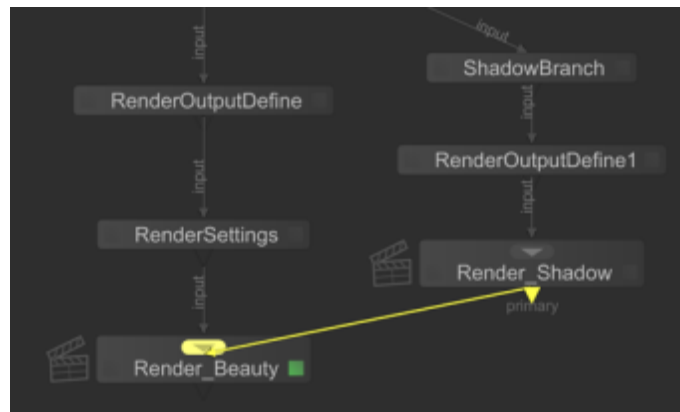
NOTE: PRMan 17 currently supports arrays with a maximum of 4 elements.



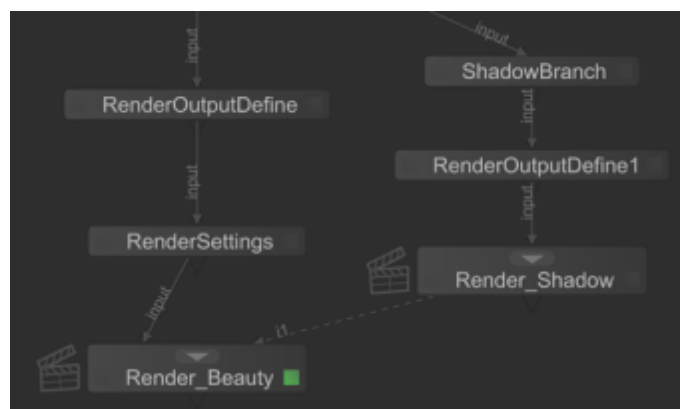
NOTE: PRMan 17 currently does not support string arrays.

Setting up Render Dependencies

Some renders may require another render to be completed first, for instance the generation of a shadow map. You can set dependencies between Render nodes by connecting the output from the Render node that needs to be run first to the large connector at the top of the other Render node.



Dependencies are shown with a dashed line.



Managing Color Within Katana

As well as communicating with one or more renderers, Katana also reads in image data from a number of different formats. Managing the color of the data within Katana is accomplished through the OpenColorIO standard, originally developed by Sony Pictures Imageworks.

A typical workflow within Katana involves:

1. Reading in the images from various formats, such as DPX, TIFF, or OpenEXR.
2. Converting those images into the scene-linear colorspace.

This is handled automatically by Katana as long as the filenames use the OpenColorIO naming scheme. Files should use a suffix denoting the file's colorspace, for instance: `beautypass_Inf.exr` (for a 32-bit linear file). For further details, see the OpenColorIO standard at:

<http://opencolorio.org/>

3. Rendering within the scene-linear colorspace.
4. Compositing and manipulating the images in scene-linear colorspace.



NOTE: Compositing with image data that has not been converted yields inconsistent results.

5. Viewing the scene-linear image data through a device-specific look-up-table (LUT) in the **Monitor** tab. The LUTs can include additional manipulations to show the image data converted to film or log (or any other potential output if you have the correct LUT) so you can see the image as it would display in that target's colorspace.
6. Writing the file out, specifying the colorspace to use in the relevant node. Use the **rendererSettings.colorSpace** parameter in the `RenderOutputDefine` node for 3D renders and the **image.colorspace** parameter in the `ImageWrite` node for 2D composites. Make sure **colorConvert** is enabled in both cases.

This is a best practice guide on how to work within Katana. That said, it is perfectly possible for you to work outside the OpenColorIO standard or even manipulate your images within log or some other colorspace. However, doing so forces you to manage all image manipulations manually.

Viewing Your Renders

Viewing your renders can be done through either the **Monitor** tab or the **Catalog** tab. The **Monitor** tab is a viewer for current and previous renders. The **Catalog** tab acts as an archive for renders and imported images. Within the **Catalog** tab you can manage images by placing them into different **slots** for later comparison or reference.

When a slot is active (its slot number is displayed beneath the **Front** image in the **Monitor** tab), new renders are placed at its top. If the current slot's top image is not locked, it is replaced by any new renders. If the top image is locked, a new render is placed above it in the slot.

Using the Monitor Tab

If more than one **Monitor** tab is open, Katana always sends the preview render results to the first **Monitor** tab that was opened in a scene. If another instance of the **Monitor** tab is opened, the image rendered in the first instance is shown, or if nothing was rendered, then nothing is shown in the preview. Any consequent renders are shown only in the first instance of the **Monitor** tab, and any additional tabs opened cannot be considered the primary instance.

Toggling Whether the Monitor Tab is Maximized

Press **Ctrl+Spacebar**, double-click on the tab name, or hold the mouse pointer over the borders of the **Monitor** tab (outside of the image display area) and press **Spacebar**.

Switching the Front and Back Images

Hold the mouse pointer inside the image display area of the **Monitor** tab and press **Spacebar**.

Changing Which Catalog Slot to Use

Press the number that corresponds to the slot, for instance **3**,

OR

Change the current **Front** image using the **Catalog** tab, see [Changing the Catalog Renders Displayed in the Monitor tab](#).

Viewing the Catalog from Inside the Monitor Tab

Press the **Tab** key (pressing the **Tab** key again returns to the **Monitor** view).

Changing the Image Size and Position

There are numerous ways to get the image to the right size and location within the **Monitor** tab.

Moving the Image Around the Monitor Tab

Middle-click and drag.

Fitting an Image to the Monitor Tab

At the top of the **Monitor** tab, select **[Current display ratio]** (for instance **1.23 : 1**) > **Frame Display Window** (or press **F**).

Viewing the Image at a 1:1 Ratio

Select **[Current display scale]** > **Reset Viewport** (or press **Home**).

The image changes size so the displayed image is one image pixel to one screen pixel, the bottom left of the image moves to the bottom left of the **Monitor** tab.

Changing the Size of the Image Within the Monitor Tab

To change the displayed image size:

Scroll the **mouse-wheel** up to zoom in (or press **+**) or scroll the **mouse-wheel** down to zoom out (or press **-**).

The image size changes by a factor of two, for example: 1:8, 1:4, 1:2, 1:1, 2:1, 4:1, 8:1. The change is reflected in the display scale at the top of the tab.

OR

Alt+middle-click and drag (drag right to zoom in, drag left to zoom out).



TIP: Katana zooms in and out around the location of the cursor.



NOTE: In many Linux windows managers, the **Alt** key is used by default as a mouse modifier key. This can cause problems in 3D applications where **Alt** is used for camera navigation in 3D environments.

You can use key mapping to assign the mouse modifier to another key, such as the  (**Super** or **Meta**) key, but the method changes depending on which flavor of Linux you're using. Please refer to the documentation on key mapping for your particular Linux distribution for more information.

Overlay Masking

The **Monitor** tab supports the application of custom overlay masks with different framing options, based on a supplied configuration file. The masks allow for semi-transparent region rendering and, when this configuration file is provided, it populates the **Monitor > Display > Masks** menu, so that you can choose from a dropdown of pre-configured overlays. Providing masks for use in Katana allows you to apply a mask on a specified frame size in the Monitor.

To set up a configuration file so that you can choose overlays from the **Monitor > Display > Masks** menu, place a **monitor_mask.xml** file in your **KATANA_RESOURCES** path.

The **K** keyboard shortcut can be used to toggle the mask, and **Alt+K** cycles through the available masks.

Formatting an XML File

An example for a properly formatted **.xml** files for use with Katana masks can be found under **\$KATANA_ROOT/plugins/Resources/Examples/monitor_mask.xml**. This file is loaded into Katana so that, by default, two monitor masks are available: **Mask Aspect + Safe Areas** and **Mask Aspect (no labels)**.



TIP: For more technical users, the information below on further formatting instructions for an XML file, may come in useful.

Masks should be defined in XML using a mask element:

```
<mask name="myMask" window="(x1,y1,x2,y2)"/>
...
</mask>
```

The window defines the mask dimensions in pixels. By default, masks (and their drawings) are scaled to fit the Display Window of the viewed image. If the aspect ratio of the mask definition does not match that of the image, the image is anchored to the lower-left corner of the mask, and the overlay scaled such that it covers the longest edge of the image. This scaling can be disabled in the **Display > Masks** menu in the monitor.

Within each mask, there must be one or more drawable elements:

```
<rect window="(x1,y1,x2,y2)"
    fillColor="(r,g,b,a)"
```

```

outlineColor="(r,g,b,a) "
outlineStippleSize="px"
outlineWidth="px"
holdoutWindow="(x1,y1,x2,y2) "
labelColor="(r,g,b,a) "
labelSize="f" />
<line p1="(x,y) " p2="(x,y) " lineColor="(r,g,b,a) " lineWidth="px" />;

```

In this example:

- x, y are pixel coordinates.
- r, g, b, and a are all normalized color components between 0-1.
- f is a multiple on the drawn size of text (best experiment to find the right size, but its a scale on 24pt text @72dpi as per FTGL).
- px is a GLLineWidth relative to the mask window definition.
- rect elements require the window attribute and one or more of fillColor or ourlineColor. Other attributes are optional.
- line elements require p1 and p2.

Once selected, these are drawn on top of renders displayed in the monitor.

Changing How to Trigger a Render

By default, you have to manually start a render by:

- right-clicking on a node and selecting one of the render options in the dropdown menu,
- using one of the menu options under **Render**,
- clicking the **2D Update Mode** on the **Monitor** tab.



NOTE: Using the **2D Update Mode** only works on previously rendered 2D nodes, and not the currently-viewed 2D node.

There are three different ways of triggering a render update. You can select your preferred method from the **2D Update Mode** on the **Monitor** tab. The options are:

- **Manual** - changes to materials, lights, or geometry transformations don't trigger a render update. To have the changes take effect, click the **Trigger 2D Update** button.
- **Pen-Up** - changes to materials, lights, or geometry transformations trigger a render update only when the mouse button is released or a parameter change is applied.
- **Continuous** - changes to materials, lights, or geometry transformations, including some manipulations in the **Viewer** tab, continuously trigger a render update.

2D Update Mode	Indicators
Manual mode	
Pen-up mode	
Continuous mode	

There is also a **3D Update Mode** on the **Monitor** tab UI, used for live rendering 3D nodes. For 3D updates, please see the information on Live Rendering in the [Rendering Your Scene](#) chapter.

To change when Katana starts a render:

- Select **Render** > **Manual Render** to only start a render manually.
- Select **Render** > **Pen-Up Render** to start a render when you release the mouse after changing a parameter or the current time.
- Select **Render** > **Drag Render** to start a render while you are changing a parameter or the current time.



TIP: These options are also available at the top of the **Monitor** tab.

Changing the Displayed Channels

To change the displayed channel:

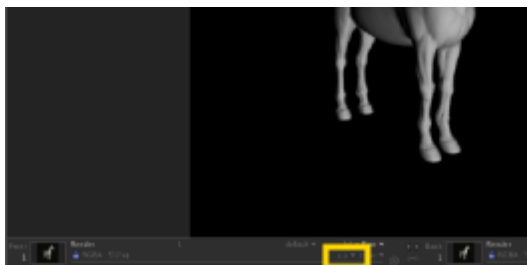
1. Click the channel display dropdown (labeled **Color** by default) towards the bottom of the **Monitor** tab.
2. Select the channel to display:
 - **Color** (or press **C**)
 - **Luma** (or press **L**)
 - **Red** (or press **R**)
 - **Green** (or press **G**)
 - **Blue** (or press **B**)
 - **Alpha** (or press **A**)



TIP: If you are viewing a channel other than the color channel, press the key that corresponds to that channel to toggle back to color. For instance, click **R** once to view the red channel, click **R** again to go back to the color channel.

Changing How the Alpha Channel is Displayed

The alpha channel menu is located next to the color display menu at the bottom of the **Monitor** tab.



Toggling Premultiply in the Monitor Tab

Select **[Alpha display] > Premultiply**.

Displaying the Alpha Channel as an Overlay

It is possible to display the alpha channel as an overlay (either as a mask or a matte).

Using the alpha channel menu the overlay is set to one of three states:

- **Mask** - The area of the image with no alpha channel becomes the overlay color.
- **Matte** - The area of the image with an alpha is overlaid with the overlay color.
- **None** - No alpha overlay is displayed.

Changing the Color Used for Alpha Overlays

1. Select **[Alpha display] > Overlay: Color**.

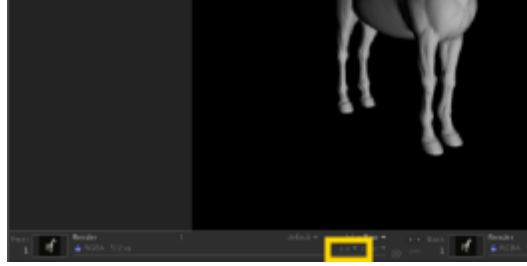
The color picker dialog displays.

2. Select a color with the color picker.
3. Press **OK**.

Selecting Which Output Pass to View

When more than just the primary pass is output during an interactive render, you can view all the outputs within the **Monitor** tab. To view outputs other than the default (primary) pass, select the output from the outputs dropdown towards the bottom of the **Monitor** tab.

By default it is **default** or **primary** (depending on the render settings).



For details on setting up multiple outputs, see [Defining an AOV Output](#). For more on sending those outputs to the **Monitor** tab, see [Previewing Interactive Renders for Outputs Other than Primary](#).

Using the Catalog tab

The **Catalog** tab acts as an archive for your renders. It has a number of slots where you can place each render. Each slot acts as a stack, you replace the top render in the stack by starting a new render. If the top item is locked, any new renders become the new head of the stack.

For more on changing which slot to use, see [Using the Monitor Tab](#) on page 186.

Viewing the RenderLog for a Catalog Entry

The **RenderLog** output for renders during this session of Katana are saved as part of its catalog entry. Catalog entries saved with a project do not include their **RenderLog**.

To view a **Catalog** entry's **RenderLog** output:

Click its thumbnail.

The entry becomes the **Front** render in the **Monitor** tab and its **RenderLog** entry is displayed within the **RenderLog** tab.

Removing Renders from the Catalog

To remove all unlocked images from the **Catalog**:

Select **Edit > Flush Unlocked Images** (from within the **Catalog** tab).

To delete the selected images from the **Catalog**:

1. Select the image(s) to delete.
2. Select **Edit > Delete Selected Images** (or press **Delete**).
3. If the images are locked, confirm deletion by clicking **Accept**.

To clear the entire **Catalog**:

1. Select **Edit > Clear Catalog**.
2. Click **Delete** to confirm.

Changing the Catalog Renders Displayed in the Monitor tab

To change the **Front** and **Back** images within the **Monitor** tab:

- left-click a thumbnail to make it the **Front** image, or,
- right-click a thumbnail to make it the **Back** image.

Manipulating Catalog Entries

Moving Catalog Entries from One Slot to Another

To move entries, middle-click and drag.

Toggling the Lock Status of a Render

To toggle the lock status, click  /  next to the image's thumbnail. Locked images are not overridden by a subsequent render to the same slot.

Toggling Whether an Image is Saved Within this Catalog

To toggle the save status:

Click  /  next to the image's thumbnail.

The first time the icon is pressed a file is saved to the directory specified by the KATANA_PERSISTENT_IMAGES_PATH environment variable (if not set, it defaults to /tmp/katana_persist). To add a prefix to the filename, use the KATANA_PERSISTENT_IMAGES_PREFIX environment variable.

Changing the Region of Interest (ROI) to Match the ROI of the Render

1. Right-click to the right of the renders thumbnail.
2. Select **Adopt Render ROI**.

Changing the Current Frame to Match the Frame of the Render

1. Right-click to the right of the renders thumbnail.

2. Select **Adopt Frame Time**.

Selecting the Node the Catalog Render was Generated From

1. Right-click to the right of the renders thumbnail.
2. Select **Find in Node Graph**.

Regenerating the Thumbnail Within the Catalog Tab

1. Right-click to the right of the renders thumbnail.
2. Select **Regenerate Thumbnail**.

Creating a Copy of a Catalog Item

1. Right-click to the right of the renders thumbnail.
2. Select **Duplicate Catalog Item**.

Making a Comment for a Catalog Render

- Type under the comment heading in the same row as the relevant thumbnail.

OR

- If the image is the current **Front** or **Back** image, click  at the top of the **Monitor** tab and type the comment in the **Front** or **Back** field.

Importing an Image or File Sequence to the Catalog

1. Select **File > Import Image / Sequence** (from the **Catalog** tab).
The **File Browser** displays.
2. Select whether you want an individual frame or a sequence by toggling **Sequence Listing**.
3. Select the image or sequence to import.
4. Click **Accept**.

Toggling the Lock for New 2D Renders

Click the checkbox marked **Lock 2D**.

When ticked, any new renders automatically have the lock icon. Being locked prevents the image being overridden by a subsequent render to the same slot.

Changing the Catalog View

By default the **Catalog** tab displays renders under their respective slot. It is also possible to view the renders in order of when they were rendered.

To change the **Catalog** tab to a **Slot** centric view:

Click **Slot View** in the upper right corner of the tab.

To change the **Catalog** tab to a **Time** centric view:

Click **Time View** in the upper right corner of the tab.

Using the Histogram

Katana comes equipped with a **Histogram** tab for checking RGBA levels within an image.



NOTE: The **Histogram** tab works in conjunction with the Pixel probe in the **Monitor** tab. To view anything within the **Histogram** tab you must have a point or area selected with the probe.

The image's channels are plotted with the value along the horizontal axis and the count for that value along the vertical axis. The top histogram matches the **Front** image and the bottom histogram matches the **Back** image.

Viewing the Count at a Particular Value

Click anywhere within either histogram.

The RGBA channel's count for that value displays towards the top of each histogram. The format of the display is <value> : <red count> <green count> <blue count> <alpha count> .

Changing the Colorspace Used for Plotting Values Within the Histogram

Click the **colorspace** dropdown and select one of the provided options - these options come from the current OpenColorIO profile. For more on OpenColorIO, see [Managing Color Within Katana](#).

Changing the Scale of the Plotted Values in the Y Axis

Enter a new value within the **vScale** field.

Toggling a Channels Display Within the Histogram Tab

Click the letter that represents the channel at the top right of the tab.



TIP: Along the bottom of each histogram a colored dot shows the lowest and highest value for each channel displayed.



NOTE: The plotted value does not necessarily correspond to an actual value. For instance, the range for the **Inf** colorspace within the **Histogram** tab is 0 to 1023 whereas the actual values are 32-bit floating point. Katana maps the colorspace values to a range for display purposes.

Viewing the Pixel Values of the Front and Back Images

Turning on the Pixel Probe

Click  or press . (full stop).

The pixel probe toolbar displays.

Changing the Colorspace for the Displayed Pixel Values

Select from the top dropdown in the pixel probe toolbar.

Changing What Type of Pixel Value You Want Displayed

Click the lower dropdown in the pixel probe toolbar and select:

- **ave** - the mean average of the area selected.
- **min** - the lowest value for each channel from the area selected.
- **max** - the highest value for each channel from the area selected.
- **stdDev** - the standard deviation of the area selected.

Changing Where the Pixel Probe Samples

- **Ctrl**+click to change the sample center point.
- Click  to sample an area and  to change back to sampling a point.
- Click and drag the center point.
- Click and drag the vertical bar to move the sample's center left and right.

- Click and drag the horizontal bar to move the sample's center up and down.
- When sampling an area, click and drag the bounding border lines to change the area's bounding rectangle.

Using the Pixel Probe to Identify Geometry

If you are viewing an image render in the **Monitor** tab and you want to identify geometry, you can use the pixel probe to do this:

1. In the **Monitor** tab, turn on the pixel probe following the instructions in [Turning on the Pixel Probe](#).
2. Pick a pixel on the object in the Monitor.
Katana's internal 'id' channel mapping identifies the geometry that this color originates from.
3. Click on the scene graph locations widget  that appears next to the geometry name in the pixel probe toolbar, and click **Select in Scenegraph** from the dropdown menu.
The scene graph location is highlighted in the **Scene Graph** tab.



NOTE: For more information on the **Monitor** tab, refer to [Using the Monitor Tab](#) on page 186.

Turning Off the Pixel Probe

Click  or press . (full stop).

The pixel probe toolbar is hidden.

Comparing Front and Back Images

If you need to compare the **Front** and **Back** images, for instance: to see how changes are affecting an image or that colors are consistent across shots, you can use the swipe feature within Katana. There are two types of swiping within Katana, line swipe (where a line acts as a curtain from one image to the next) and a rectangle swipe (where a bounding rectangle displays the **Back** image inside the rectangle and **Front** image outside).

Using the Swipe Line Feature

Select **[Swipe menu] > Swipe Line**.

The swipe line handle displays in the **Monitor** tab and the names for the **Front** and **Back** images become **A** and **B** respectively. You can:

- Click and drag the center of the handle to move its origin.
- Click and drag the lines either side of the handles center to change the swipe angle.

Using the Swipe Rectangle Feature

Select **[Swipe menu] > Swipe Rect**

The swipe rectangle displays in the **Monitor** tab and the names for the **Front** and **Back** images become **Outside** and **Inside** respectively. You can:

- Click and drag the center of the handle to move its origin.
- Click and drag the bounding box lines to change the swipe rectangle.

Turning on a Red/Cyan 3D Mix Between the Front and Back Images

Select **[Swipe menu] > Red/Cyan 3D**.

Turning Off Swiping (and Any Swipe Menu's Red/Cyan 3D Mix)

Select **[Swipe menu] > Swipe Off**.

Toggling 2D Manipulator Display

Some 2D nodes, such as **Transform2D**, provide a manipulator within the **Monitor** tab. It is possible to toggle the display of these manipulators. To toggle the display of 2D nodes' manipulators, click  or .

Underlaying and Overlaying an Image

The **Monitor** tab within Katana has the ability to overlay or underlay an image with the current render. The underlay and overlay can be composited with either the **Over** or **Add** function.

Displaying the Underlay and Overlay Controls

Click .

The **Underlay/Overlay** toolbar is added to the **Monitor** tab.

Adding an Image to the Underlay or Overlay Fields

Middle-click and drag from either the **Front/Back** images in the **Monitor** tab or one of the renders from the **Catalog** tab. The checkbox toggles on and the Underlay/Overlay function becomes active.

Turning Off the Underlay or Overlay Composition

Uncheck the checkbox to the left of the field name.

Removing an Image from the Underlay or Overlay Fields

Click  to the right of the image.

Changing the Compositing Function Used

1. Click the dropdown on the left of the toolbar.
2. Select the compositing function, the options are **Add** or **Over**.

Removing the Underlay/Overlay Toolbar

Click .

Rendering a Region of Interest (ROI)

To reduce render time while making changes, you can render a smaller section of the image - this section is called a region of interest (ROI). The region of interest is only used for interactive renders and is ignored when performing a Disk Render, or Disk Render with Dependencies.

Switching On Region of Interest Rendering

Select **[ROI menu] > ROI on** or **ROI on (visible)**.

Switching Off Region of Interest Rendering

Select **[ROI menu] > ROI off** or **ROI off (visible)**.



NOTE: If the region of interest's bounding rectangle is visible, you can change the bounds by dragging the edges.

Using the Viewer

Overview

The **Viewer** tab provides one or more 3D windows into the scene described by the scene graph. Only locations that are exposed within the **Scene Graph** tab are represented in the **Viewer** - the exception being pinned locations. For more on pinning a location see [Pinning a Location or Locations](#).

You can interactively modify parameters, on some nodes contributing to a scene graph location, using **Manipulators** within the Viewer. The manipulators available vary depending on the scene graph location selected, and the nodes that created it. For more on this see [Using Manipulators](#). It is also possible for additional manipulators to be implemented by your studio using the Viewer Manipulator API. Consult the developer documentation and example code for further details.

In **Shaded (raw)** and **Shaded (filmlook)** modes the Viewer uses OpenGL lights and shaders, which are distinct from the Arnold or PRMan lights and shaders used for final rendering. The OpenGL lights and shaders are added to existing Arnold or PRMan light and material nodes. For more on this see [Assigning a Viewer Material Shader](#), and [Assigning a Viewer Light Shader](#).

Changing How the Scene is Displayed

You can change the 3D scene within the **Viewer** to suit your needs, the computer's specifications, or a scene's specific demands.

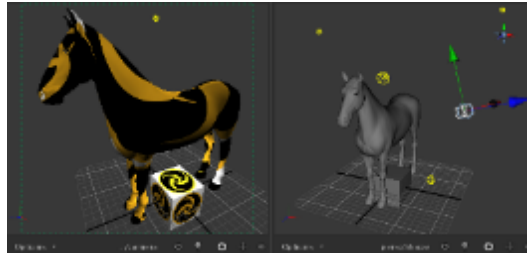
Changing the Layout

The **Viewer** tab can be split into multiple tabs allowing multiple views of the same scene. Each tab has its own settings for shading, and lighting modes.

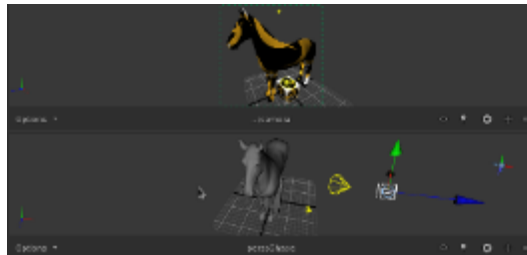
Splitting the Viewer Tab into Multiple Tabs

To split the **Viewer** tab, select **Layout > ...**:

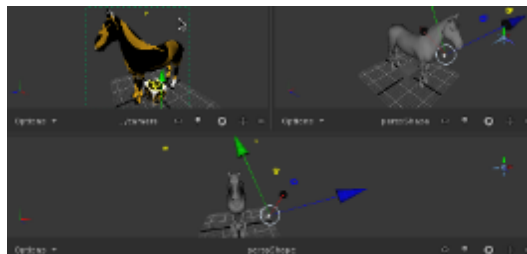
- **Single Tab** - A single tab takes up the whole **Viewer**, this is the default.
- **Two Tabs Side-by-Side** - This displays two tabs split vertically, sitting side-by-side.



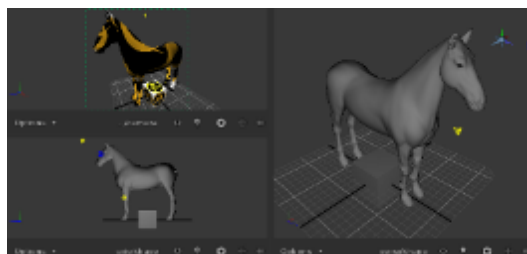
- **Two Tabs Stacked** - This displays two tabs split horizontally, one above the other.



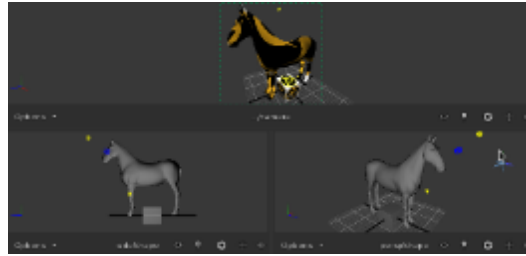
- **Three Tabs Split Top** - This displays three tabs, one large on the bottom, and two more split vertically above.



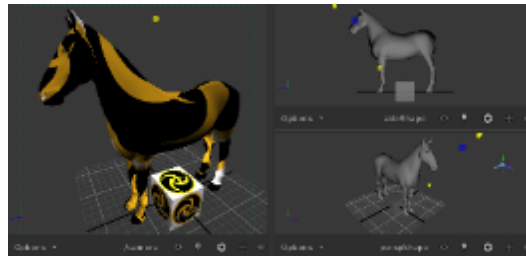
- **Three Tabs Split Left** - This displays three tabs, one large on the right, and two more split horizontally on the left.



- **Three Tabs Split Bottom** - This displays three tabs, one large on the top, and two more split vertically below.



- **Three Tabs Split Right** - This displays three tabs, one large on the left, and two more split horizontally on the right.



- **Four Tabs** - This displays four tabs.

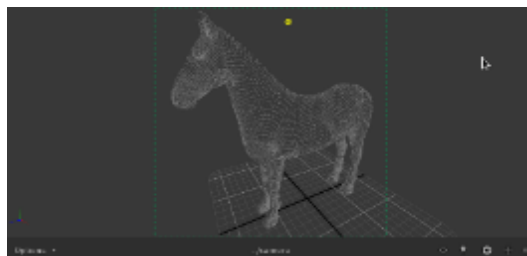


You can change each tab to have a different view of the scene graph data. The current view is either an object within the scene - such as a camera or light - or a **Viewer** camera. A **Viewer** camera is not a part of the scene graph and cannot be used outside the **Viewer**. Four **Viewer** cameras are created by default (**persp**, **top**, **front**, and **side**). Additional Viewer perspective cameras are created by clicking on the **New persp view** button in the Viewer **Look Through Lights and Cameras** menu. For more on this see [Changing what you Look Through](#).

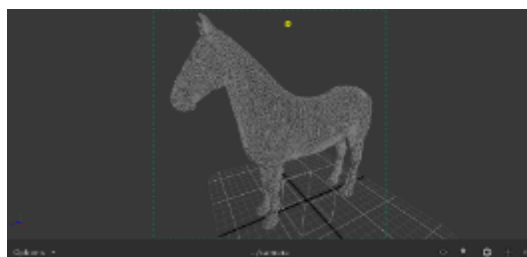
Changing the Overall Viewer Behavior

To change the overall shading model, select **Display > ...**:

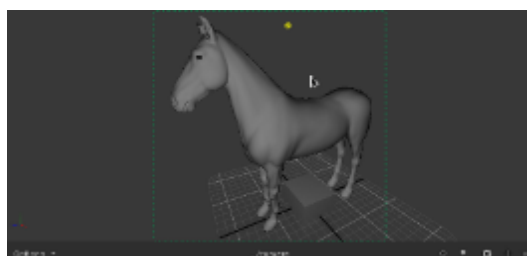
- **Points** - This displays the current 3D scene with each vertex (or control point for a NURBS patch) as a point.



- **Wireframe** (or press **4**) - This displays the current 3D scene with each edge (or surface curve for a NURBS patch) as a line.



- **Simple Shaded** (or press **6**) - This displays the current 3D scene with a very simple shader, ignoring scene lights and shadows.



- **Shaded (raw)** - This displays the current 3D scene with each object using its Viewer shader (or the default if none is assigned). Adding a Viewer shader is covered in [Managing Color Within Katana](#)



- **Shaded (filmlook)** (or press **5**) - This is identical to the **Shaded (raw)** shading model but applies an adjustment designed to approximate the **filmlook** OpenColorIO LUT. For more information on OpenColorIO within Katana see [Managing Color Within Katana](#).



NOTE: As **Shaded (raw)** and **Shaded (filmlook)** use OpenGL shaders, and not the shaders used for the final render the **Viewer** can display a drastically different look to your final render depending on how closely the OpenGL shaders matches the production shaders.

Using Flush Caches

Katana stores scene graph information in a series of caches, including caches for resolved shaders and lights.

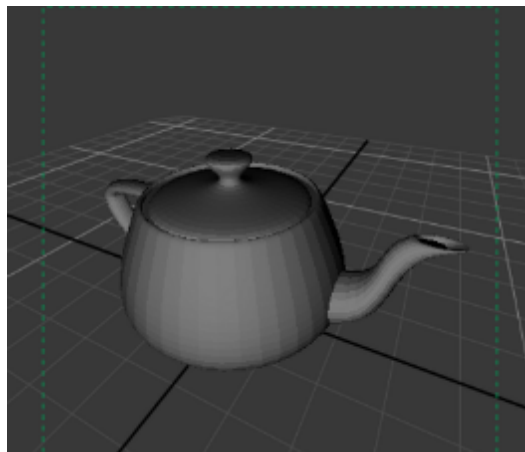
Selecting **Util > Flush Caches**, or clicking on the **Flush Caches** button  forces Katana to step through the scene graph, resolving shaders and lights. This in turn clears and update the Viewer cache.

Assigning a Viewer Material Shader

The Viewer is OpenGL based, so for materials to display in the **Viewer**, they must have an OpenGL **Viewer shader**. It is this **Viewer shader**, not any Arnold or PRMan shader that the Viewer shows when in **Shaded (raw)** or **Shaded (filmlook)** modes. For example, create a primitive, assign an Arnold or PRMan shader, and observe the **Viewer** output in **Shaded (raw)** mode.

1. Create a primitive using a Primitive Create node.
2. Create an Arnold or PRMan material using a Material node.
3. Assign the material to the primitive using a MaterialAssign node.
4. Add a **KatanaSpotlight** using a GafferThree node, then position it.
5. Change the Viewer to **Options > Shaded (raw)** mode.

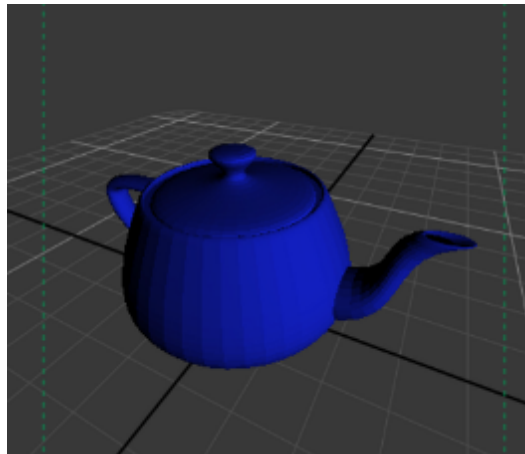
Without a Viewer shader assigned, the primitive in the Viewer defaults to a gray Lambert.



Add a Viewer shader to your material, and observe the **Viewer** output in **Shaded (raw)** mode.

1. Open the recipe described above.
2. Edit the parameters of the **Material** node.
3. Click **Add shader**, and select **Surface** from the dropdown list.
4. Select **KatanaPhong** as the Viewer shader type.
5. Edit the diffuse color of the **KatanaPhong** shader.
6. Change the viewer to **Options > Shaded (raw)** mode.

The viewer displays the **Viewer shader** added to the material node.



NOTE: To have Viewer shaders mirror production shaders, it's advisable to link by expression common parameters, such as diffuse and specular color, or texture file path.

Displaying Textures in the Viewer

If the texture maps used in your PRMan or Arnold final shaders are in the form of **.tx** or **.tex** files, you can show these in the Viewer, provided they have the file suffix **.tx**, rather than **.tex**. In addition, the Viewer can render RGB and RGBA image formats, such as **.tif**, **.png**, and **.jpg**. For example, create a primitive, assign it a Viewer shader material, and map the **Texture** parameter of the Viewer shader material to an image file.

1. Open the recipe created in [Assigning a Viewer Material Shader](#).
2. Edit the parameters of the **Material** node.
3. Expand the parameters of the **viewerSurfaceShader** of type **KatanaPhong** you added previously.
4. Expand the **Texture** parameter field.
5. Right-click on **filename** and select **Wide editor**.
6. Type in the path to your texture file.
7. Select **Util > Flush Caches**, or click on the **Flush Caches** button .

Observe the results in the Viewer.



NOTE: For you to apply texture color maps, your Viewer Surface shader must be either of the supplied **KatanaPhong**, or **texture** types, or a custom Viewer shader that supports this feature.

Assigning a Viewer Light Shader

As supplied, the Katana Viewer supports a single type of OpenGL spotlight, which corresponds to the Katana Spotlight final render light. For lights of types other than Katana Spotlight to display in the **Viewer**, they must have a **Viewer light shader** assigned. It is this **Viewer light shader**, not the Arnold or PRMan light shader, that the Viewer displays when in **Shaded (raw)** or **Shaded (filmlook)** modes. For example, create a primitive, and assign it a material that also has a Viewer material shader. Create a light, then observe the Viewer output in **Shaded (raw)** mode.

1. Open the recipe created in [Assigning a Viewer Material Shader](#).
2. Select the GafferThree node, then the light you created earlier, and go to its **Material** tab.
3. Click **Add Shader** and select **light** from the dropdown menu.
4. Select one of the following lights as the shader type: **KatanaBasicPointlight**, **KatanaBasicSpotlight**, or **KatanaSpotlight**.
5. Change the Viewer to **Options > Shaded (raw)** mode.

The Viewer shows the Viewer light, and changes to the Viewer light update in the Viewer.

Changing Which Lights to Use

To change the lighting used for the Shaded (raw & filmlook) shading models:

- Select **Display > Lighting > Off** - This removes all lights from the **Viewer**.
- Select **Display > Lighting > Selected Lights** (or press **8**) - All selected lights contribute to the lighting in the **Viewer**.
- Select **Display > Lighting > All Lights** (or press **7**) - All lights within the scene contribute to the lighting in the **Viewer**.

Changing Shadow Behavior

To change whether shadows are used for the Shaded (raw & filmlook) shading models:

- Select **Display > Shadows > Off** - No shadows from lights are used in the **Viewer**.
- Select **Display > Shadows > Selected Lights** - All selected lights create shadows for the lighting in the **Viewer**.
- Select **Display > Shadows > All Lights** - All lights create shadows for the lighting in the **Viewer**.

Changing the Anti-Aliasing Settings

To change the anti-aliasing for lines and points:

- Select **Display > Smoothing > Off** - Anti-aliasing is not applied to either points or lines.

- Select **Display > Smoothing > Points** - Toggles point anti-aliasing in the **Viewer**.
- Select **Display > Smoothing > Lines** - Toggles line anti-aliasing in the **Viewer**.

Changing how Proxies Are Displayed

To change how proxies are displayed:

- Select **Display > Proxies > Bounding Box** (or press **Ctrl+B**) - Only proxy bounding boxes are displayed.
- Select **Display > Proxies > Geometry** (or press **Ctrl+G**) - Only proxy geometry is displayed.
- Select **Display > Proxies > Both** (or press **Ctrl+Shift+G**) - Both proxy geometry and proxy bounding boxes are displayed.



NOTE: If no proxies have been associated with the geometry, bounding boxes are not automatically calculated.

Setting Different Display Properties for Some Locations

You can override the currently selected display method for a number of locations within the **Viewer** tab using the **ViewerObjectSettings** node. To change how one or more locations are displayed:

1. Add a **ViewerObjectSettings** node to the recipe at some point before the current view node.
2. Select the **ViewerObjectSettings** node and press **Alt+E**.

The **ViewerObjectSettings** node becomes editable within the **Parameters** tab.

3. Assign the scene graph geometry locations of the objects to influence to the **CEL** parameter. See [Assigning Locations to a CEL Parameter](#) for more on using **CEL** parameter fields.
4. Set any changes to how the locations should be displayed using the node's parameters. The following display options can be set:

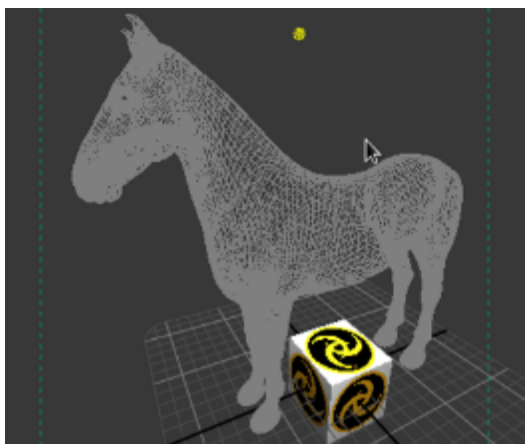
(in the **drawOptions** parameter grouping)

- **hide** - when set to **Yes**, the selected locations are hidden (as are their children).
- **fill** - changes how the location is displayed, as **points**, as a **wireframe**, as a **solid**, or to use the **Viewer's** default render type (**inherit**).
- **light** - changes the lighting model to either the simple shaded model (**default**) or the current viewer shader (**shaded**). When set to **shaded**, the default viewer shader is used if none is currently assigned. Changing an object or lights viewer shader is done in the same way as assigning any other shader. See [Creating a Material](#).
- **smoothing** - changes whether locations have aliasing. You can have aliasing on **points**, **lines**, or **both** (or it can be turned **off**).
- **windingOrder** - sets whether the location should be drawn with a **clockwise** or **counterclockwise** winding order. The correct value depends on how the imported geometry was exported from its original package.
- **pointSize** - when displaying the location using the points display type, this option sets the size of the points.

For example, the following image shows two objects, both of which have the same PRMan and Viewer Shader material applied. The Viewer is in **Shaded (raw)** mode, so each object is lit, and textured.



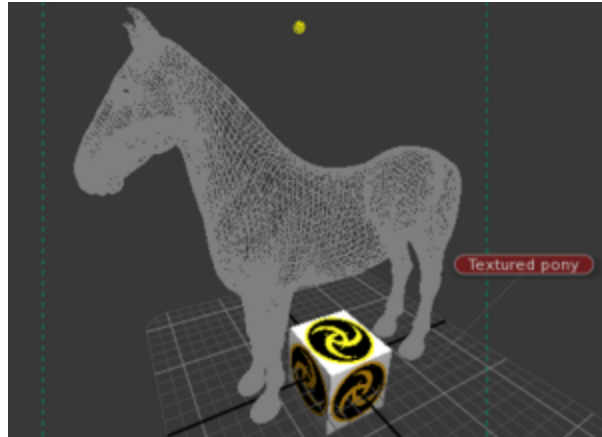
The next image shows the same scene, with the addition of a ViewerObjectSettings node. The CEL in the node points to the pony, and the **drawOptions** parameters **fill** setting is set to **Wireframe**. The Viewer's draw mode for the pony object is overridden.



(in the **annotation** parameter grouping)

- **text** - displays this text with the location.
- **color** - the background color of the annotation text.
- **pickable** - when set to **No**, you can no longer select the object in the **Viewer**.

For example, the following image shows two objects, both of which have the same PRMan and Viewer Shader material applied. There is a ViewerObjectSettings node overriding the Viewer mode for the pony, and showing a label with the contents of the **text** field in the **annotation** parameters. The background color of the label is taken from the **color** field in the **annotation** parameters.



Changing the Viewer Behavior for Locations that are Selected

By default the **Viewer** tab highlights (with a white wireframe) the location(s) that are currently selected.

To change the way Katana displays selected locations:

Select **Display > ... while selected > ...**



NOTE: Any display changes made only affect locations while they are selected.

Changing the Viewer Behavior While Dragging

For some scenes with complicated geometry or lighting it may make sense to lower the display quality while dragging geometry or lights around the scene.

To change the way Katana displays the scene while dragging:

Select **Display > ... while dragging > ...**



NOTE: Any settings within this menu override the default display behavior while something within the viewer is being dragged.

Changing the Background Color

The background color for the tab can be changed to make the scene easier to read, to reduce eye fatigue, or to better match the background color when rendered.

To change the background color, select **Display > Background Color > ...** :

- **Black** (or press **T**)

- **Gray** (or press **Alt+T**)
- **White** (or press **Shift+T**)

Overriding the Display Within a Specific Tab

You can change the shading settings in a specific tab to reduce or improve the quality. This is useful when positioning a light in one tab while viewing the effect in another.

To change a **Viewer** tab's display, use the **Options** menu in the bottom left of the tab. Each menu option corresponds to a similar one under the **Display** menu and acts as an override. To remove any override use **No Change**.

Selecting Within the Viewer

You can use standard selection behavior within the **Viewer**.

Action	Behavior
Click	Selects the first object below the mouse.
Drag	Selects all objects within or touched by the marquee.
Shift +click	Selects an object if it is not selected, deselects it if it is.
Shift +drag	Selects any object within the marquee that is not selected, deselects it if it is.
Ctrl +click	Deselects the first object below the mouse.
Ctrl +drag	Deselects everything within the marquee.

Stepping Through the Selection History

Katana tracks what is selected in the scene graph. You can step backward and forward through this selection history.

To step backward through the selection history:

Select **Selection > History Backward** (or press **Backspace**).

To step forward through the selection history:

Select **Selection > History Forward** (or press **Shift+Backspace**).

Changing the View Position

You can change which object you are viewing through and that object's position and orientation. This makes light and camera positioning easy.

Viewport Movement

To change the view's current position and orientation:

Shortcut	Action
Alt +left-click and drag	Tumbles the view around its center of interest.
Alt +middle-click and drag	Tracks the view.
Alt +right-click and drag	Dollies the view forward (drag right) and back (drag left).



NOTE: Looking through a location with no xform attribute does not allow you to move the object within the Viewport. To enable transformation of a scene graph location, add a **Transform3D** node and assign the location to the node's **path** parameter.



NOTE: In many Linux windows managers, the **Alt** key is used by default as a mouse modifier key. This can cause problems in 3D applications where **Alt** is used for camera navigation in 3D environments.

You can use key mapping to assign the mouse modifier to another key, such as the  (**Super** or **Meta**) key, but the method changes depending on which flavor of Linux you're using. Please refer to the documentation on key mapping for your particular Linux distribution for more information.

Changing what you Look Through

The view from a Viewport comes from either a light or a camera. You can change the view to a different light or camera to make placement easier or to help with composition.

Selecting the View from the Camera and Light List

1. Click the text at the bottom of the Viewport (such as **perspShape**).
This brings up a list of available lights and cameras.
2. Filter the list to find the camera or light you want. To filter the list you can:

- Uncheck the **Cameras** checkbox to remove cameras from the list.
 - Uncheck the **Lights** checkbox to remove lights from the list.
 - Type text into the **Filter** field to only display items that contain the text.
3. Select the required light or camera from the list.

Alternatively, you can also:

1. Click the text at the bottom of the Viewport (such as **perspShape**).
This brings up a list of lights and cameras.
2. Click **New persp view** to look through a new perspective camera.



NOTE: The camera and lights displayed in the filter list are populated in four ways:

- Cameras from the **globals.cameraList** at the **/root/world** location.
 - Lights from the **lightList** attribute at the **/root/world** location.
 - The default four cameras (persp, top, front and side) along with any new cameras created with the **New-persp view** button in the filter list.
- The current render camera (such as set with the **RenderSettings** node).



NOTE: Cameras with a scene graph location can be identified by the  icon by their name in the filter list.

Selecting the View from the Camera List

1. Click  to bring up the camera list.
2. Type text into the **Filter** field to only display cameras that contain the text.
3. Select the camera to look through from the list.

Alternatively, you can also:

1. Click  to bring up the camera list.
2. Click **New persp view** to look through a new perspective camera.

Selecting the View from the Light List

1. Click  to bring up the light list.
2. Type text into the **Filter** field to only display lights that contain the text.
3. Select the light to look through from the list.

Selecting the View from a Scene Graph Location

1. Select the scene graph location to look through.
2. Click .



TIP: Text entered into the **Filter** field of the view selection dialogs may contain some basic regular expression patterns, such as ranges [a-z].

Looking Around the Viewport by Offsetting and Overscanning

Looking around the Viewport without actually moving the camera is especially useful when a camera has been brought in from another package - representing a camera track for instance - and you don't want to change its position or orientation.

To look around inside the Viewport:

1. Click  to bring up the pan/zoom toolbar.
2. To make changes to the current view:
 - Type in the **hOff** field to pan left (negative value) or right (positive value).
 - Type in the **vOff** field to pan up (positive value) or down (negative value).
 - Type in the **overscan** field to zoom in (value between zero and one) or out (value above one).
3. Click **Reset** to restore defaults.



TIP: All three text fields can be scrubbed by dragging on their names.

While you have the toolbar up the **Pan-zoom active** warning text is displayed in the top left corner of the Viewport.

When **hOff**, **vOff**, or **overscan** values change from their defaults, Katana displays a warning icon  on the left of the toolbar.

Changing What is Displayed Within the Viewport

Customizing the **Viewer** or individual viewports to only display the information you need can help speed up your workflow.

Hiding and Unhiding Objects Within the Scene

Objects within the **Viewer** can be hidden from view.

Hiding Object(s) Within the Viewer

1. Select the object(s) within the **Viewer** (or select the locations within the **Scene Graph** tab).
2. Select **Selection > Hide** (or press **H**).

Elements are hidden is displayed in all viewports when one or more objects are hidden.

Making All Hidden Objects Visible

Select **Selection > Unhide All** (or press **U**).

Changing the Subdivision Level of a Subdivision Surface

Subdivision surfaces (Subds) are a form of polymesh that allows greater detail to be defined in certain areas of a mesh while keeping the rest of the mesh at a rough lower level.

To change the displayed level of a subdivision surface:

1. Select the object(s) you want to change.
2. Select **Selection > Subd Level ...** (or press **0, 1, 2, or 3**).



NOTE: Use higher levels of subdivision with caution as they can be expensive to calculate.

Toggling Grid Display

Katana displays a **Grid** to help you get a sense of scale, the origin's location, and the orientation of the XZ plane.

To toggle displaying the Grid:

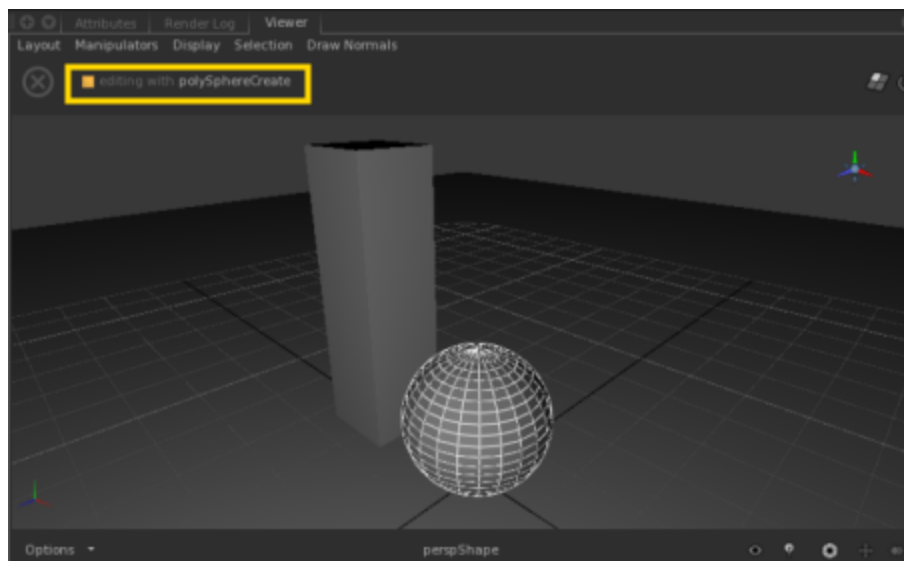
Select **Display > Grid** (or press **G**).

Using Manipulators

Manipulators provide a visual way for you to edit parameters of applicable nodes in the node graph that contribute to the selected scene graph location. Each manipulator is only available for locations created by nodes that have a parameters corresponding to that manipulator. For example, if you select a location in the scene graph, you can only move it with a translation manipulator in the Viewer if the nodes that create it have parameters that map to an interactive transform, such as a light created by a GafferThree or LightCreate node, a primitive created by a PrimitiveCreate node, or a mesh with a Transform3D node targeted to its scene graph location.

If a manipulator is grayed out for a particular scene graph location, it means that Katana is unable to find Parameters on a Node in the Node Graph with the ability to affect the relevant attributes at that location.

When an applicable scene graph location is selected, the name of the node that receives primary manipulations is displayed at the top of the **Viewer** tab.

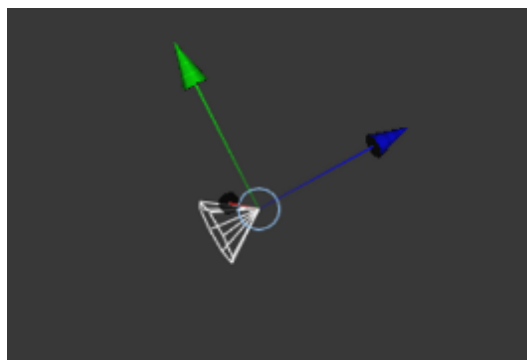


Toggling Manipulator Display

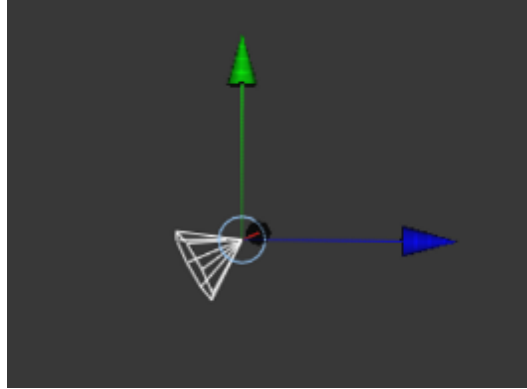
Select **Display > Manipulators >** (or press **' (Backtick)**) to toggle Manipulator display on or off.

Select **Manipulators > No Transform Manipulator** (or press **Q**) to toggle off any enabled Transform Manipulators.

Select **Manipulators > Translate** (or press **W**) to toggle the local space Transform Manipulator on or off.

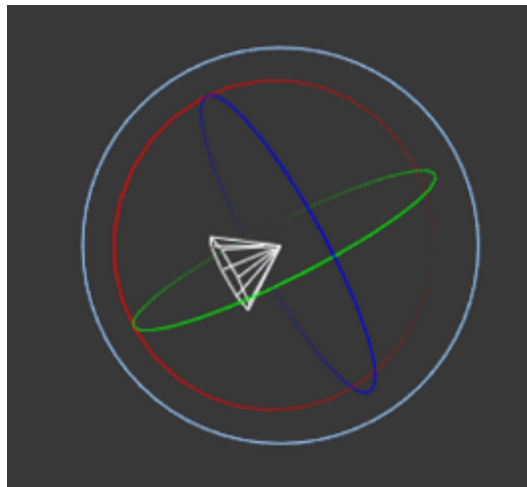


Select **Manipulators > Translate (world)** (or press **S**) to toggle the world space Transform Manipulator on or off.

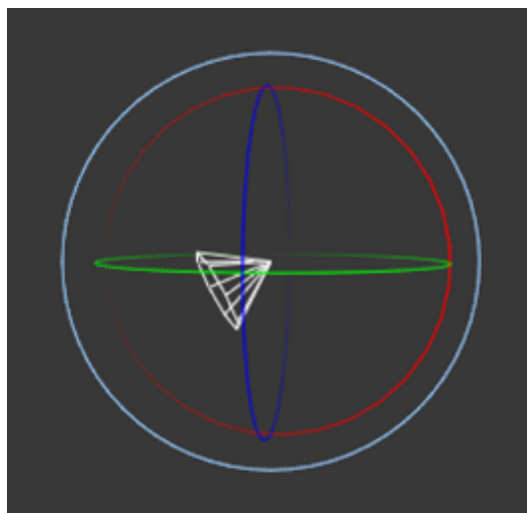


Select **Manipulators > Translate Around COI** (or press **S**, or **Tab**) to toggle the translate around center of interest Manipulator on or off.

Select **Manipulators > Rotate** (or press **E**) to toggle the local space rotation manipulator on or off.



Select **Manipulators > Rotate (world)** (or press **D**) to toggle the world space rotation manipulator on or off.



Select **Manipulators > Rotate Around COI** (or press **E**, or **Tab**) to toggle the rotate around center of interest manipulator on or off.



NOTE: Selecting **Rotate Around COI** sets the rotate Manipulator to the position of the center of interest of the selected object, oriented to the local space of the selected object.

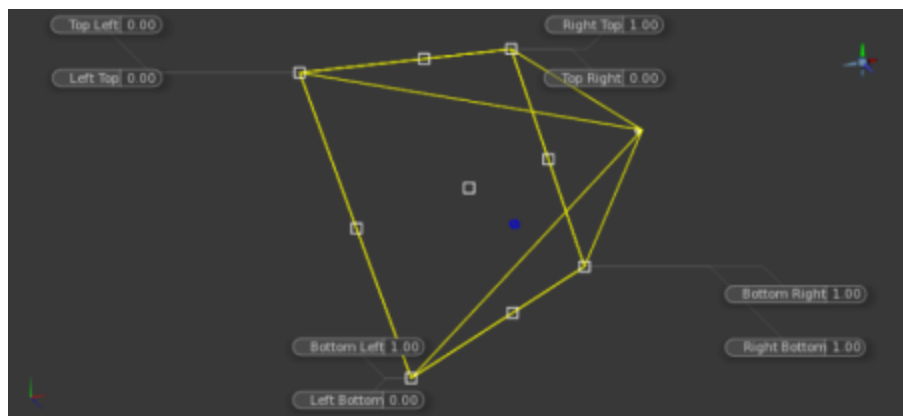
Toggling Annotation Display

Some manipulators have **Annotations** to display parameter values. You can turn these **Annotations** off.

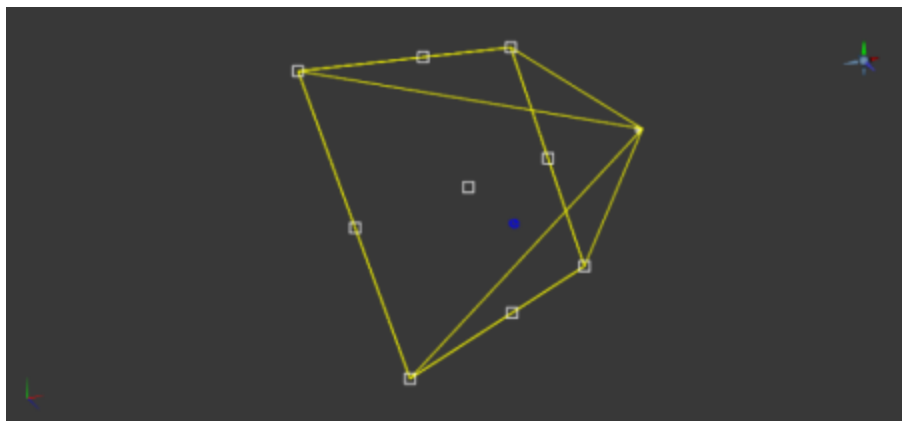
To toggle displaying Annotations for manipulators:

Select **Display > Annotations** (or press **Shift+~**).

For example, selecting a light of type **KatanaSpotlight**, then selecting **Manipulators > Barn Door** shows the interactive Manipulators for the lights barn door parameters, along with **Annotations** showing the barn door parameter names, and their values.



If you select **Display > Annotations** (or press **Shift+~**), the **Annotations** are removed, but the **Manipulator** remains.



Toggling the Heads Up Display (HUD)

Within Katana each **Viewer** tab has its own axis orientation guide in the bottom left corner. The default perspective camera (and any other perspective cameras made with the **New persp view** button) has a manipulator in the top right corner to change the cameras position to a view axis, or three quarter view, centered on the current selection. You can hide these features.

To toggle the display of the Heads Up Display (HUD):

Select **Display > HUD**.

Displaying Normal Information Within the Viewer

Katana gives you the ability to display object normals.

To toggle normal display within the **Viewer** select **Display > Normals** (or press **N**).

To change the normals display length:

- select **Draw Normals > Scale ...**, or
- enter the required normal size in **viewerSettings.normalsDisplayScale** in the **Project Settings** tab.

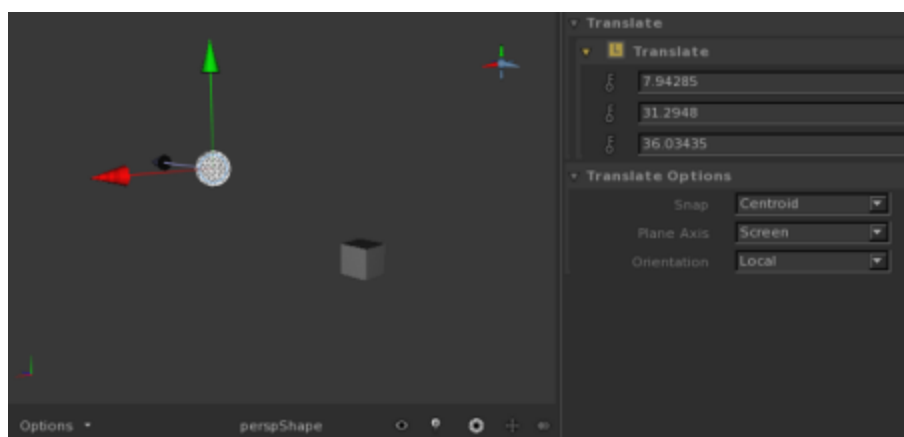
Transforming an Object in the Viewer

Using translation Manipulators you can move, rotate, and scale objects within the Viewer

With the **Quick Editor** you can change the translation Manipulator's coordinate systems, plane axis, and whether the Manipulator snaps. You can also manually type values for any parameters represented by the Manipulator. To display the **Quick Editor**, make sure you have nothing selected, select the menu option **Layout > Show Quick Editor** (or press the **A** key). Then select an object, and a Manipulator to modify that Manipulator's settings.

For example, activate **Snap** on the Translate Manipulator, and snap one primitive onto another.

1. Create a **Primitive** using a **Primitive Create** node.
2. Add another **Primitive**, using a **Primitive Create** node.
3. Create a **Merge** node, and connect each **Primitive** to it.
4. Move one **Primitive** away from the other.
5. Select **Layout > Show Quick Editor** (or press the **A** key).
6. Select the **Primitive** you want to snap onto the other.
7. Select **Manipulators > Translate** (or press **W**).
8. Expand the **Translate Options** group, and change the **Snap** option to **Centroid**.
9. Drag one **Primitive** over the other, and it **Snaps** to the **Centroid** of the second **Primitive**.



NOTE: Depending on your screen resolution, you may need to expand the size of the **Quick Editor** to see all of the available options. To expand the **Quick Editor** window, left-click and drag on its border.

Translating an Object in Its Local Coordinate System

1. Select the object to translate.
2. Select **Manipulators > Translate** (or press **W**).

Translating an Object in the World Coordinate System

1. Select the object to translate.
2. Select **Manipulators > Translate (world)** (or press **S**).

Rotating an Object in its Local Coordinate System

1. Select the object to rotate.
2. Select **Manipulators > Rotate** (or press **E**).

Rotating an Object in the World Coordinate System

1. Select the object to rotate.
2. Select **Manipulators > Rotate (world)** (or press **D**).

Scaling an Object

1. Select the object to scale.
2. Select **Manipulators > Scale** (or press **R**).

Removing All Transform Manipulators

Select **Manipulators > No Transform Manipulator** (or press **Q**).

Manipulating a Light Source

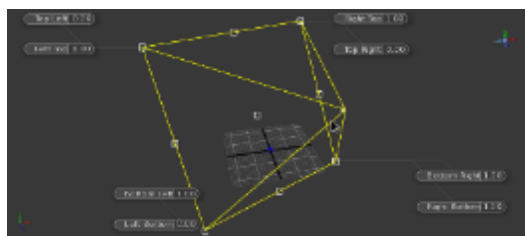
In addition to the translation and rotation Manipulators covered in [Toggling Manipulator Display](#), Katana offers Manipulators to interactively adjust light parameters. Some parameters easily changed with a Manipulator are: barn doors, the cone angle, decay regions, and its Gobo. The light itself must have the required parameters in order to use the manipulator, for instance you cannot use the barn door manipulator on a light which does not support barn doors (the menu option would not be displayed).



NOTE: Custom lights need to both support the function represented for a Manipulator to show, and have matching parameter names.

Manipulating the Barn Doors for a Light

1. Select the light to manipulate.
2. Select **Manipulators > Barn Door**.
3. Move one or more of the nine square manipulators to the desired position.



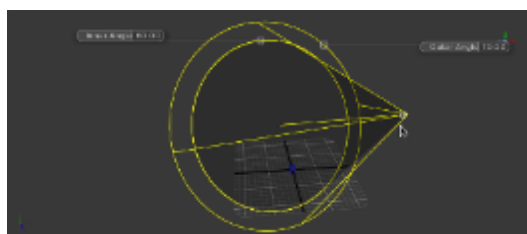
NOTE: Each parameter is defined by a value between 0 and 1.

Changing a Light's Center of Interest

1. Select the light to manipulate.
2. Select **Manipulators > Center of Interest**.
3. Move the circular manipulator to where you wish the light to point.

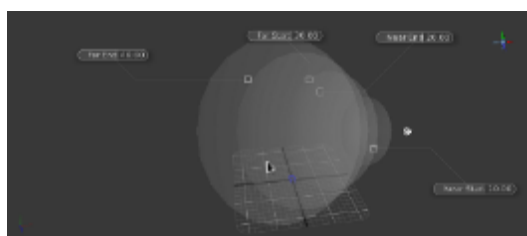
Changing a Light's Cone Angle

1. Select the light to manipulate.
2. Select **Manipulators > Cone Angle**.
3. Move the two manipulators to change the inner and outer cone angles.



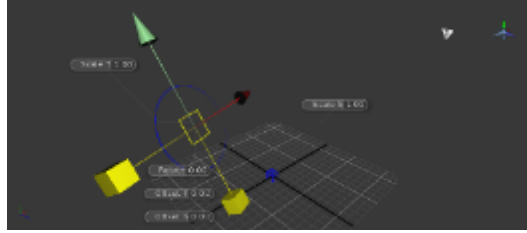
Changing a Light's Decay Regions

1. Select the light to manipulate.
2. Select **Manipulators > Decay Regions**.
3. Move the four manipulators to change the distance of the decay regions from the light source.



Scaling and Positioning a Lights Gobo

1. Select the light to manipulate.
2. Select **Manipulators > Slide Map**.
3. Move the Gobo with the translate manipulators.
4. Scale the Gobo with the scale manipulators.



Rotating the Light Around Its Center of Interest

1. Select the light to manipulate.
2. Select **Manipulators > Rotate Around COI**.
3. Use the rotate manipulator to move the light around the center of interest.

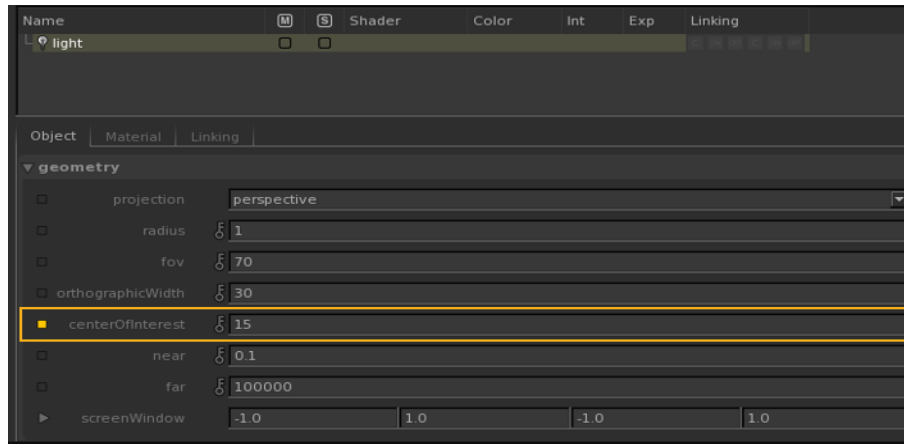
Moving the Light While Keeping it Pointed at Its Center of Interest

1. Select the light to manipulate.
2. Select **Manipulators > Translate Around COI**.
3. Move the light with the translate manipulator.
The light remains pointed towards its center of interest.

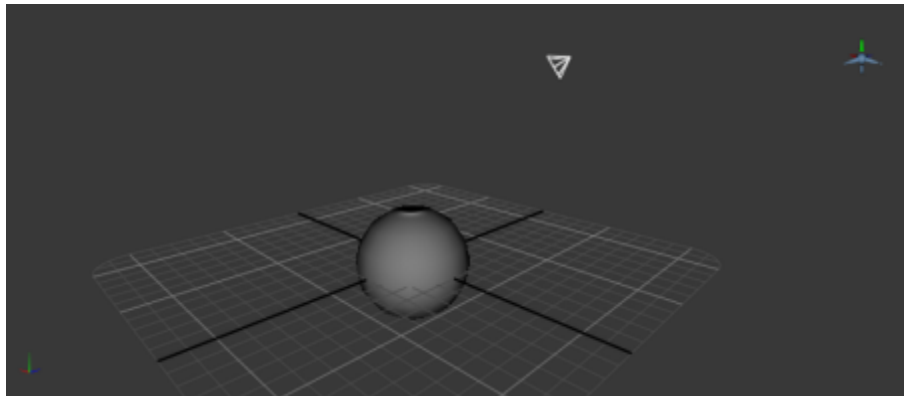
Positioning a Light so Its Specular Highlight is at a Specific Point

Use **Manipulators > Place Specular** to position a selected light at the Center of Interest (COI) distance from a selected point on a surface, with the light's Z axis aligned with the surface Normal at that point. For example, create a primitive sphere, and a spotlight, then use **Place Specular** to position and orient the light.

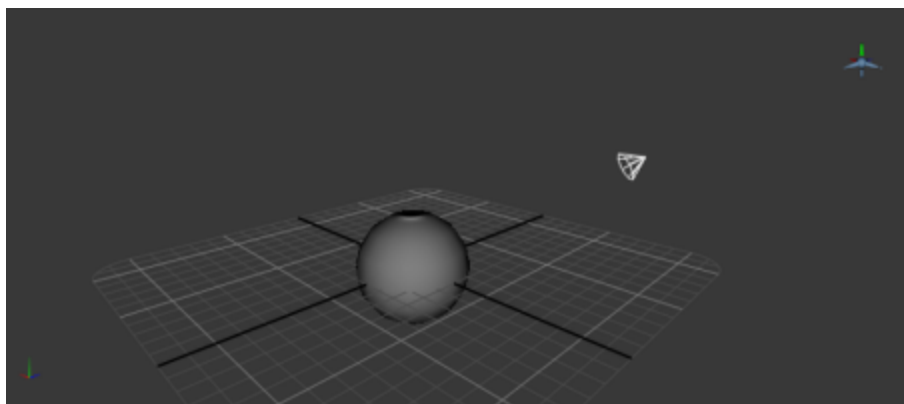
1. Create a Primitive using a PrimitiveCreate node, and leave the type as the default **Sphere**.
2. Create a GafferThree node and add a light of type **KatanaSpotlight**.
See [Creating a Light Using the GafferThree Node](#) for information on how to add lights to your scene using a GafferThree node.
3. Edit the **centerOfInterest** parameter of your light, to a distance of your choice.



4. Create a Merge node and connect the Primitive and GafferThree nodes to it.
5. Expand the scene graph at the Merge node and view your Katana Spotlight, and Primitive in the Viewer.
See [Viewing the Scene Graph](#) for information on viewing the scene graph.

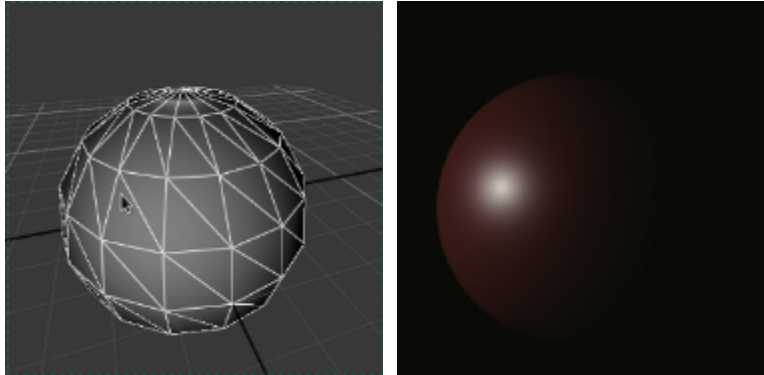


6. Select your light in the **Scene Graph** tab, or by clicking on its icon in the **Viewer** tab.
7. In the **Viewer** tab, select **Manipulators > Place Specular**.
8. Click on the surface of the sphere, where you want the specular highlight to display.
Katana moves the selected light to a point coincident with the chosen point on the surface, aligned with its Z axis parallel to the surface Normal, and translated the COI distance along Z.



9. If you set the Viewer to look through your camera, you'll see that - with **Place Specular** active - you can click on the sphere in the Viewer where you want the specular highlight to appear, then render, and the light position and orientation adjust accordingly.

See [Changing what you Look Through](#) for information on how to set the Viewer's look through object.



TIP: To move forward through the light manipulator list, press the **Tab** key. To move backward through the list, press **Shift+Tab**. To have no light manipulator, press **Shift+Q**.

Checking UVs

Overview

The **UV Viewer** tab allows you to examine the UVs of the currently selected object in the scene graph, both for subdivision surfaces and polygonal meshes. An asset's UVs are not usually manipulated inside Katana, instead, the UVs should be included when the asset is published.

Bringing up the UV Viewer Tab

You can display the **UV Viewer** tab in one of two ways:

- Select **Tabs > UV Viewer** to display the **UV Viewer** in its own floating panel, or
- In the top left corner of one of the existing panes, click  and select **UV Viewer** to add the tab to that pane.

Navigating in the UV Viewer Tab

Moving around inside the **UV Viewer** is done in a similar manner to moving in the **Node Graph** tab.

Panning

Panning with the Mouse

Middle-click and drag the mouse pointer over the grid within the **UV Viewer**. The UV space moves with your pointer.

Zooming

There are multiple options for zooming into or out from the UV grid.

Zooming In

Move your mouse pointer over the area you want to zoom in on, and then:

- Press **+** (**Plus** key) repeatedly until the **UV Viewer** displays the UV space at the desired scale.
- **Alt**+left/right-click and drag right.
- Scroll up with the mouse wheel.

Zooming Out

Move your mouse pointer over the area you want to zoom out from, and then:

- Press **-** (**Minus** key) repeatedly until the **UV Viewer** displays the UV space at the desired scale.
- **Alt**+left/right-click and drag left.
- Scroll down with the mouse wheel.

Framing

To change the framing within the **UV Viewer**, you can:

- Press **A** to frame all the UVs.
- Press **F** to frame the currently selected UVs.
- Select the coordinate space from the **Frame UV** dropdown towards the top of the **UV Viewer** tab, for instance **0, 0**.

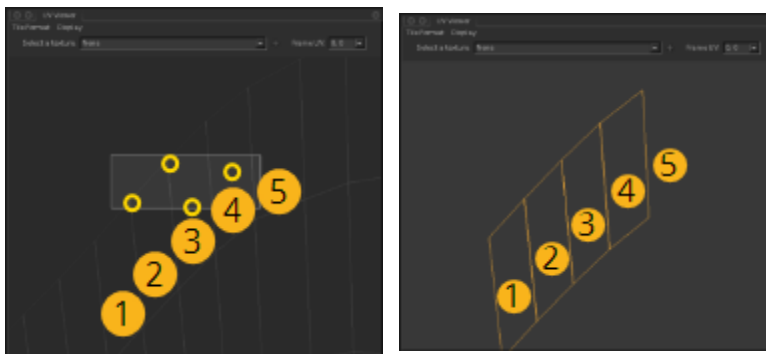
Selecting Faces

Whether creating face sets or seeing where the UV faces fall on the model, it is possible to select faces using the **UV Viewer** and then see the same faces selected in the **Viewer**. The reverse is also possible.

Selecting One or More Faces

To select one or more faces:

Left-click and drag to marquee an area. Faces with at least one UV coordinate or the face's center encompassed by the marquee are selected.



Face 1 has its top-right UV coordinate selected.

Face 2 has its top-left and top-right UV coordinates selected.

Face 3 has its top-left UV coordinate and face center selected.

Face 4 has its center selected.

Face 5 is not selected as none of its UV coordinates are selected or its center.

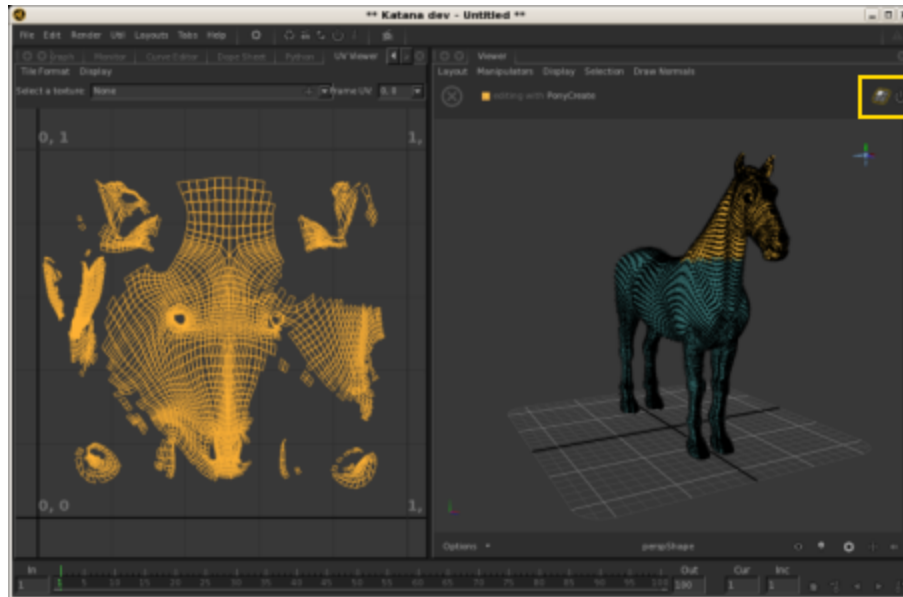
Modifying an Existing Selection

To modify the selection, you can:

- Hold **Shift** while marqueeing to toggle whether or not faces are selected.
- Hold **Ctrl** while marqueeing to remove faces from the selection.
- Hold **Ctrl+Shift** while marqueeing to append to the selection.

Viewing the Selected Faces in the Viewer

For the faces to be visible on the model, the **Viewer** must be in face selection mode. To toggle face selection mode, in the top-right corner of the **Viewer** tab, click .



Adding Textures to the UV Viewer

The **UV Viewer** automatically detects texture filename attributes on the currently selected scene graph location (which is the same as the currently selected **Viewer** object) when placed under **textures** in the location's attributes, for instance **textures.ColMap**.

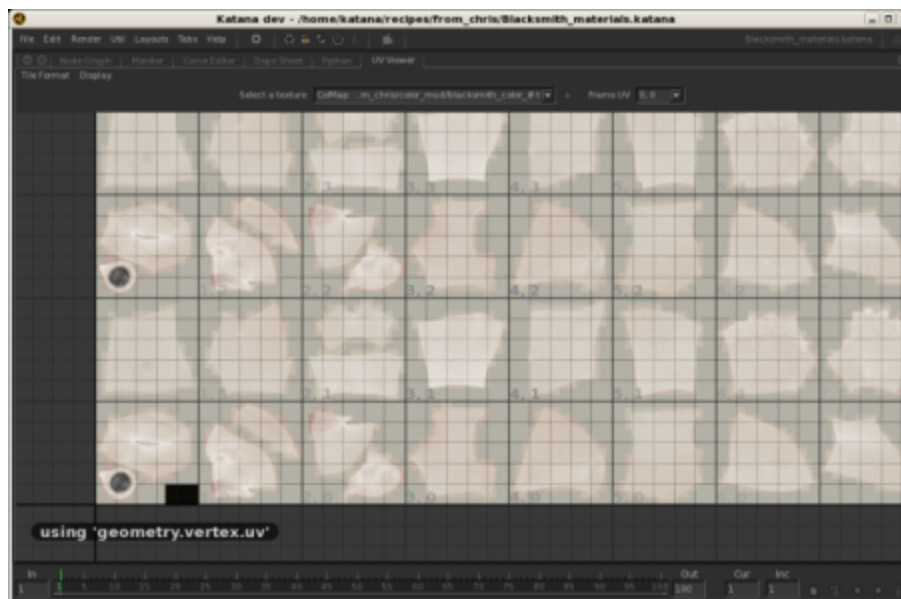


NOTE: Before trying to load multi-tile textures, make sure the correct format is selected in the **Tile Format** menu in the **UV Viewer**, for instance **UDIM**.

Loading the Textures

To load textures into the **UV Viewer**, you can:

- Select the texture name from the **Select a texture** dropdown.
- To the right of the **Select a texture** dropdown, click **+** and select the texture from the dialog.



NOTE: Valid texture formats are: **.tif**, **.exr**, **.tx**, and **.jpg**.

Using Multi-Tile Textures

The **UV Viewer** currently supports two different naming schemes for multi-tile textures, UDIM and the naming scheme used in Autodesk® Mudbox™.

Working with UDIMs

UDIM values identify the integer position of a texture or patch. Each patch represents one square of 1x1 in UV space. UDIM values are a way of representing the integer coordinates of that square, from the coordinates of its bottom-left corner in UV space. UDIMs are up to ten patches across, and any number of patches upwards. This means the U index of a patch is in the range 0 to 9, and the V index upwards from 0. To calculate the exact UDIM value for a patch, use the following formula: $1001 + u + (10 * v)$.

For example, the UDIM of the bottom-left patch, which represents the UV space region (0, 0) to (1, 1), is 1001. The next patch to the right of that has a UDIM value of 1002, and the patch directly above the bottom-left is 1011. For example, the patch representing the UV space region (2, 5) to (3, 6) has a U index of 2 and a V index of 5, so replacing the values in the formula above we get: $1001 + 2 + (10 * 5) = 1053$.

One way to load textures that use the UDIM format, is to use a filename of the form **<asset_name>_<texture_type>.#.<file_type>**, for instance **blacksmith_color.#.tx**. The exact format used depends on your asset naming scheme, file sequence plug-in, and texture type. For more on *asset management and the file sequence plug-in*, see the included *technical PDFs in the* `${KATANA_ROOT}/docs/pdf` directory.

Working with Mudbox Ttextures

Multi-tile textures exported from Mudbox have a texture name of the form **<name>_u<#>_v<#>.<file_type>**, for instance **blacksmith_u1_v2.tif**.

Multi-Tile Naming Convention Table

...				
1011	1012	1013	1014	
0,1 to 1,2	1,1 to 2,2	2,1 to 3,2	3,1 to 4,2	
(0,1)	(1,1)	(2,1)	(3,1)	
u1_v2	u2_v2	u3_v2	u4_v2	
UDIM: 1001	1002	1003	1004	
Area: 0,0 to 1,1	1,0 to 2,1	2,0 to 3,1	3,0 to 4,1	
UV Tile: (0,0)	(1,0)	(2,0)	(3,0)	...
Mudbox: u1_v1	u2_v1	u3_v1	u4_v1	

Changing the UV Viewer Display

You can customize how the **UV Viewer** displays textures, UVs, and the labels that are displayed.

Showing Faces in Isolation

To toggle whether selected faces are shown in isolation:

Select **Display > Isolate Selection**. If active, when faces are selected all other faces become hidden.

Changing Tile Labels

To change the labels for each tile:

Select **Display > Tile Position Labels > ...**

- **Off** - hides the position labels.

- **Show Tile Co-ordinates** - displays the UV space coordinates of the tile, for instance 1, 0.
- **Show Tile IDs** - displays the tile name based on its tile texture name. This is dependent on the tile format in use, for instance u1_v2 for textures saved using the Mudbox file naming format and 1002 for textures saved using the UDIM file naming format.

Displaying the Tile Resolution

To show the texture resolution for any textures:

Select **Display > Show Texture Resolution**.

Clipping Textures to Poly Faces

To only display textures when contained within a face:

Select **Display > Clip Textures to Poly**.

Using the Timeline

Using the Timeline

Katana's timeline allows you to move from one frame to another and view keyframes over the frame range.

Changing the Current Frame

To change frames in Katana, you can:

- Press the **Right Arrow** key to increment the current frame by **Inc**, or **Left Arrow** key to decrement.
- Click  to increment the current frame by **Inc**, or  to decrement.
- Click on the timeline at the relevant frame.
- Type the frame number in the field marked **Cur**.
- Press **Ctrl+Right Arrow** to jump to the next keyframe, or **Ctrl+Left Arrow** to jump back to the previous.
- Click  to jump to the next keyframe, or  to jump back to the previous.

Panning the Frame Range

To pan the current frame range, you can:

- Drag the timeline with the middle mouse button, or
- drag the scroll bar directly under the time range.

Zooming the Frame Range

To zoom into/out of an area of the frame range, you can:

- **Ctrl**+drag to select an area of the frame range, then upon release of the mouse button the timeline zooms to that range.
- Scroll up with the mouse wheel over a frame to zoom in at that point, or scroll down to zoom out.
- Press the **+** key to zoom in, or the **-** key to zoom out.

- Click  to set the range from **inTime** to **outTime** in the **Project Settings** tab.
- Press the **Home** key to set the range from **inTime** to **outTime** in the **Project Settings** tab.
- Press the **F** key to set the range to fit all keyframes on the timeline.

Changing the Frame Range In and Out Points

To change the frame range in and out points, you can:

- Press the **[** key to set the in point to the current frame, or press the **]** key to set the out point.
- Type the in frame number into the **In** field on the timeline, or type the out frame number in the **Out** field.

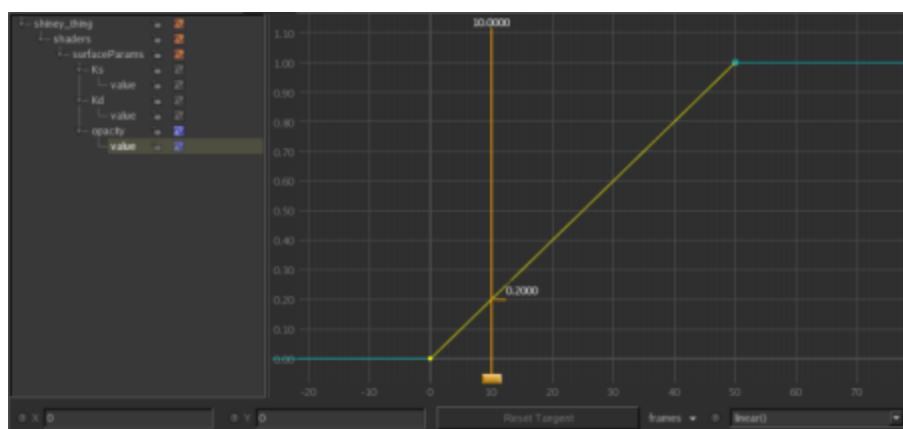
Animating Within Katana

Animation Introduction

Computer based animation owes its core concepts to the techniques employed by pencil-drawn animators since the dawn of the animation business. In order to reduce time, the lead animators of large studios would draw key poses - known as **keyframes** or **keys** - defining the extreme positions within a scene. A different animator would then fill in the poses between the keyframes using a technique called tweening, thereby creating the illusion of movement. For some scenes, **breakdowns** were created to show how the transition from one keyframe flowed to the next.

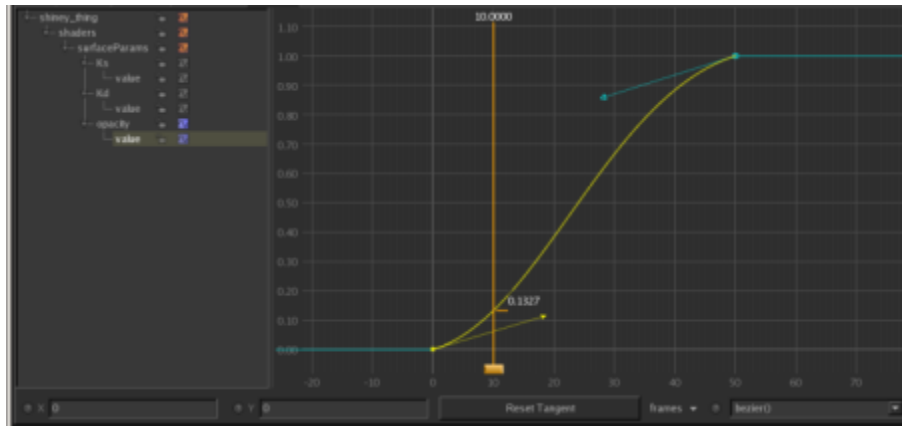
Nearly one hundred years later, this technique - known as **keyframing** - is still alive and kicking within Katana.

Katana does the animation heavy lifting by interpolating the values between keyframes. You can tell Katana how you want these **in-between** frames to be generated by specifying a **segment function**.



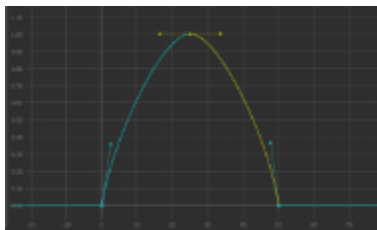
Two keyframes on frames zero and fifty with a linear segment function applied to the first.

The most versatile segment function is the **bezier** curve; it uses a mathematical formula to calculate a curve between two anchor points. Bezier curves use four points to interpolate a curve: two anchor points (these are the keyframes) and two control points.

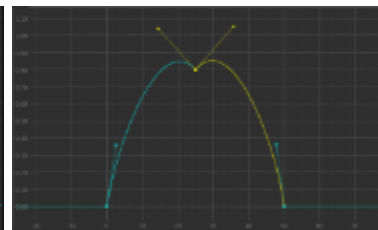


The same two keyframes with the bezier segment function applied. The arrowheads represent the location of the two control points.

A tangent and its control points control the slope of the curve around the tangent's keyframe.

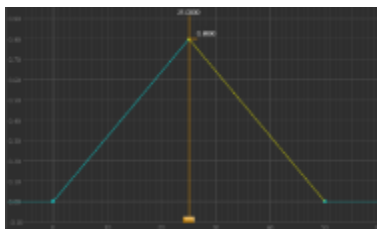


The selected control point handles, shown in yellow, form a tangent around the keyframe.

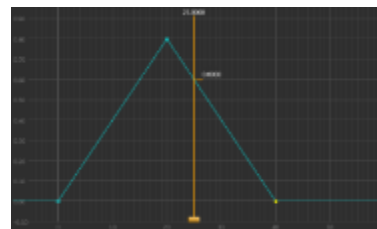


Here, a straight line between control points would not pass through the keyframe; hence the tangent is broken.

Breakdowns within the **Curve Editor** maintain the relative time between the keyframe before and the keyframe after.



Keyframes have been placed on frames 0, 25, and 50. The middle keyframe, on frame 25, has been converted into a breakdown.



Moving the third keyframe from frame 50 to frame 40, automatically moves the breakdown to frame 20.

Keyframes, breakdowns, segment functions, and tangents all combine to create a **curve** that represents how a value changes over time. A curve is plotted on a graph within the Curve Editor tab with time (in frames) along the x axis and the parameter's value plotted on the y axis. When a parameter uses a curve, its background color within the **Parameter** tab changes to green. Light green signifies that the parameter has a keyframe at the current frame; a dark green parameter signifies that the value is interpolated.



A bright green parameter signifies a keyframe on the current frame.



A dark green parameter signifies the value for the current frame is interpolated.

Setting Keys

You can set keys either manually or Katana can automatically set a key every time you change the parameter value. To have Katana automatically create keys when you enter a new value, you need to turn on **Auto Key** mode for that parameter.

Toggling Auto Key

While a parameter has the **Auto Key** icon highlighted , entering a value in the parameter field creates a new keyframe at the current frame.

To toggle **Auto Key** mode:

- Right-click on the parameter to toggle and select **Auto Key**.

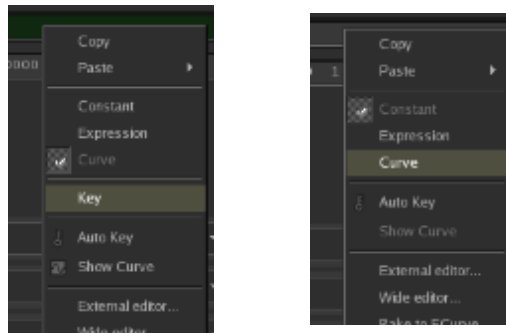
OR

- Click the **Auto Key** icon,  / , next to the parameter.

Setting Keys Manually

To set a key manually:

1. Move the **Timeline** to the correct frame.
2. Set the parameter to the desired value.
3. Right-click the parameter and select **Key**. If a key has not been set on the parameter before, select **Curve**.
Selecting Curve not only sets a key, it also converts that parameter from a Constant or Expression to a Curve.



NOTE: You can also set keys within the **Curve Editor** tab using **Insert** mode, as well as converting an interpolated value into a key. See [Setting Keys in the Curve Editor](#) and [Baking a Segment of the Curve](#) for more information.

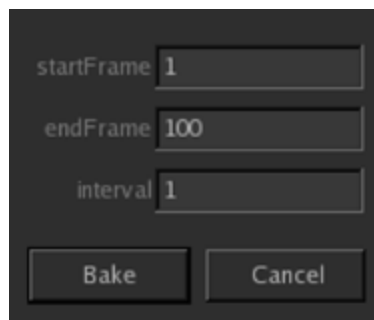
Baking a Curve

Whether from an expression or a keyframed curve, you can convert part or all of it to keyframes.

Generating Keyframes from a Curve or Expression

1. Right-click on the parameter.
2. Select **Bake to FCurve...**

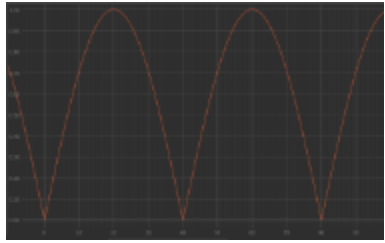
The **Bake to Curve** dialog displays.



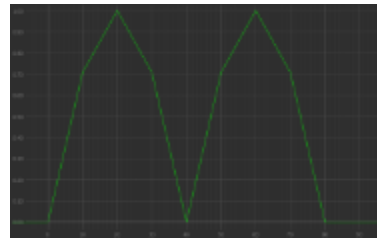
3. Change the dialog values to suit the curve you are creating. You can change the:
 - **startFrame** - the frame to start generating keys.
 - **endFrame** - the last frame to generate a key.
 - **interval** - how often to generate a key (in frames).
4. Click **Bake**.

The parameter changes from an expression to a curve and keys are generated from **startFrame** to **endFrame**.

All the newly generated keys are assigned the linear segment function.



The expression `abs(sin(frame*pi/40))` displayed in the Curve Editor.



The curve generated by baking with a startFrame of 0, endFrame of 80, and an interval of 10.



NOTE: Although most commonly used with expressions, **Bake to FCurve...** can be used to automatically generate keyframes for any type of parameter, whether it's an expression, a constant, or already a curve.

Exporting and Importing a Curve

Curves can be exported and imported.

Exporting a Curve

1. Right-click on the parameter to export.
2. Select **Export FCurve...**

Importing a Curve

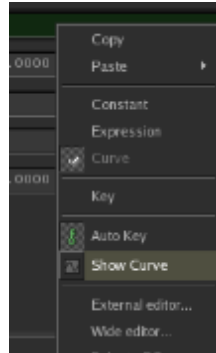
1. Right-click on the parameter to change.
2. Select **Import FCurve...**

Displaying Keyframes

You can use the **Curve Editor** tab, **Dope Sheet** tab, and **Timeline** to view and manipulate keyframes. They only show a parameter's keyframes if the parameter has the **Show Curve** icon highlighted.

To toggle the **Show Curve** icon:

1. Right-click on the parameter.
2. Select the **Show Curve** menu item.

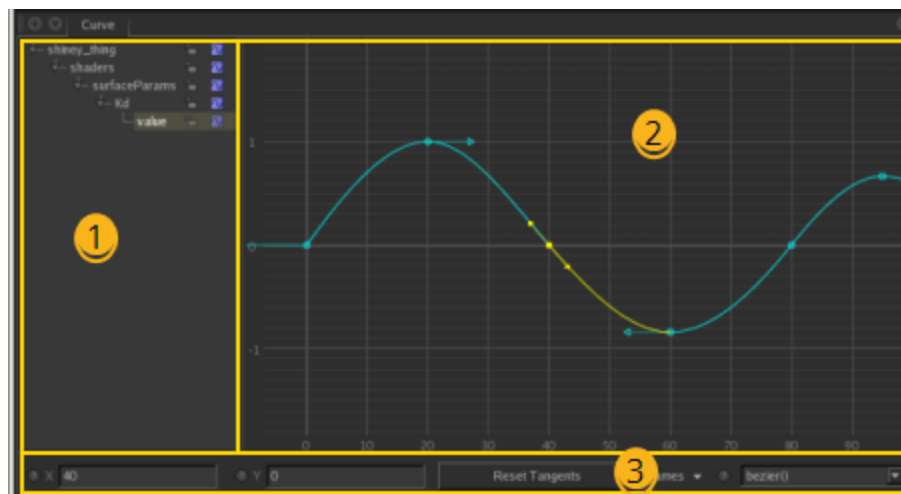


OR

Click  or  to the left of the parameter input field.

Curve Editor Overview

The **Curve Editor** is the heart of animating within Katana. Here you can move keyframes; change their segment function, tangents and weights; set breakdowns; and make any curve manipulations necessary to get the curve you need.



The **Curve Editor** is split into three areas:

1. The left-hand side is a hierarchical view of all parameters with **Show Curve** enabled.
2. The right-hand side shows these parameter values plotted over time. The parameter value range is on the left and the time frame across the bottom. This area is referred to as the **Curve Editor** graph.
3. The bottom of the **Curve Editor** has a toolbar containing ways to manipulate the keyframes.



TIP: Although the **Curve Editor** is primarily for manipulating curves, it can also be used to view the results of an **Expression**. To view an **Expression** in the **Curve Editor**, enable **Show Curve** for the **Expression** parameter.

Using the Hierarchical View

On the left of the **Curve Editor** is a hierarchical view of the curves and expressions that have **Show Curve** enabled. You can use this view to expand and collapse the parameters, lock the curves against editing, and toggle the curves that are shown in the **Curve Editor** graph.

Expanding or Collapsing a Curve

Double-click on the part of the parameter name to expand or collapse.

OR

Click  to expand or  to collapse.



NOTE: Collapsing a parameter in the hierarchical view only changes whether its children are displayed in the hierarchical view. Its only use is to keep the hierarchy more manageable.

Selecting a Curve in the Hierarchical View

Click on a parameter name to select its curve - it must be the leaf name as that corresponds to the actual parameter.



TIP: You can select more than one parameter by **Ctrl**+clicking further parameters and **Shift**+clicking to select all the parameters from your last selection to where you click.

Locking a Curve

You can lock a parameter to stop its curve from being editable within the **Curve Editor**.

Locking a Parameter and Stop it from Being Editable Within the Curve Editor

Click  within the hierarchical view in the **Curve Editor**.

Unlocking a Parameter

Click .



NOTE: Parameters that are expressions are always locked and cannot be modified within the **Curve Editor**.

Hiding and Showing a Curve

Even though a parameter has **Show Curve** selected, you may not want to display it within the **Curve Editor** graph.

To Hide a Parameter Curve Within the Curve Editor:

Click  within the hierarchical view in the **Curve Editor**.

To Show a Parameter Curve that has Previously Been Hidden:

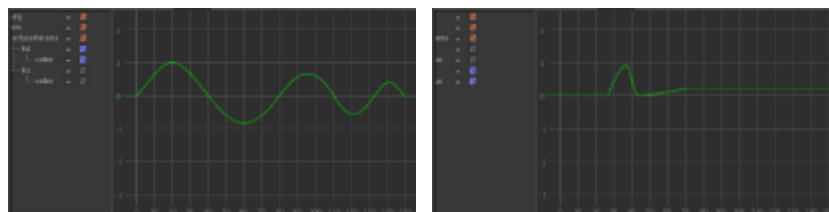
Click  within the hierarchical view in the **Curve Editor**.

Switching the Display of a Parameter's Children

When only some of the children of a parameter are shown,  is displayed.

To Switch the Display State of the Children of a Parameter Name:

Click  within the hierarchical view in the **Curve Editor**.



By clicking  the two child curves have changed their display states - one becoming hidden and the other visible.

Setting Keys in the Curve Editor

To set keys quickly and easily within the **Curve Editor**, you can use insert mode. The insert mode enables you to click on the graph at any point and insert a new key at that position.

To insert keys with insert mode:

1. Select the curve for the new keys.
2. Press the **Insert** key.
This puts you into insert mode.
3. Click a point on the graph to insert a new key at that position.
4. Repeat step 3 to insert as many keys as required.
Select a different curve within the hierarchical view to insert keys on that curve.
5. To finish adding keys and disable insert mode, press **Insert** again.

Selecting Keyframes in the Curve Editor

Selecting Keyframes

click on them,

OR

marquee drag over them.

Selecting All Keyframes for a Curve

Double click the curve.

Adding to the Current Selection

Hold **Shift** while selecting the keyframe(s).

Removing from the Current Selection

Hold **Ctrl** while selecting the keyframe(s) to remove.

Moving Keyframes in the Curve Editor

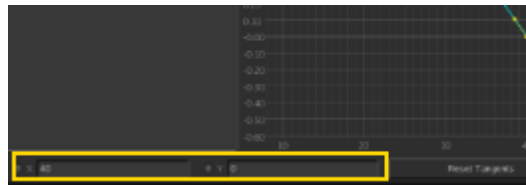
You have two ways to move keyframes within the Curve Editor: using the mouse or using the **X** and **Y** input fields.

Moving Keyframes Using the Mouse

1. Select the keyframe(s) you want to move.
2. Click-and-drag one of the selected keyframes.

Moving a Single Keyframe Using the Input Fields

1. Select the keyframe you want to move.
2. Make any changes in the input fields below the graph:
 - Enter a new frame number in the **X** input field.
 - Enter a new value in the **Y** input field.



The values entered into **X** and **Y** are absolute and not relative. For instance, entering 10 in the **X** input field, moves the keyframe to frame 10.

Moving Multiple Keyframes Using the Input Fields

1. Select the keyframes you want to move.
2. Make any changes in the input fields below the graph. All changes are relative, for instance 3 would add 3 to the current value or frame number and -3 would subtract 3 from the current value or frame number:
 - Enter a relative frame number in the **X+** input field.
 - Enter a relative value in the **Y+** input field.



Changing the Display Range in the Curve Editor Graph

Katana provides a number of ways to change the frame range and parameter value range in the Curve Editor graph.

Panning the Curve Editor Graph

Middle-click and drag within the graph area.

Panning in a Single Axis

Shift+middle-click and drag within the graph area.

Zooming the Curve Editor Graph In and Out

Use the scroll wheel, scroll up to zoom in and down to zoom out.

OR

Press **+** (**Plus** key) to zoom in or press **-** (**Minus** key) to zoom out.

Framing All the Keyframes in the Curve Editor Graph

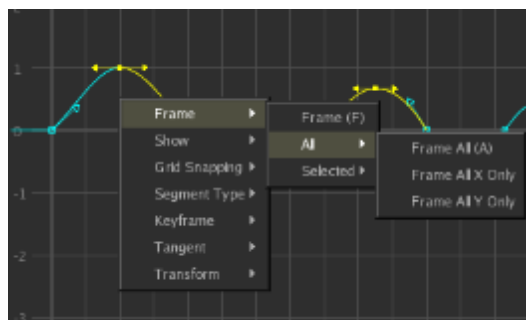
Right-click and select **Frame** > **All** > **Frame All** (or press **A**).

Framing All the Keyframes in the Curve Editor Graph in the X Axis

Right-click and select **Frame** > **All** > **Frame All X Only**.

To Frame All the Keyframes in the Curve Editor Graph in the Y Axis:

Right-click and select **Frame** > **All** > **Frame All Y Only**.



Framing the Selected Keyframes

Right-click and select **Frame** > **Frame** (or press **F**).

OR

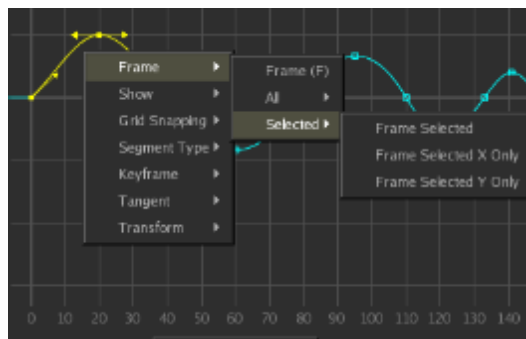
Right-click and select **Frame > Selected > Frame Selected**.

Framing the Selected Keyframes in the X Axis

Right-click and select **Frame > Selected > Frame Selected X Only**.

Framing the Selected Keyframes in the Y Axis

Right-click and select **Frame > Selected > Frame Selected Y Only**.



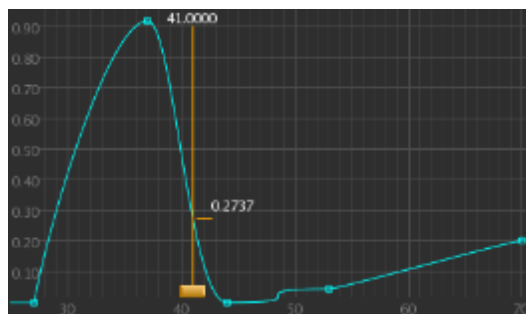
Changing Display Elements Within the Curve Editor Graph

You can display other information in conjunction with the parameter curves. Additional elements that can be displayed include: a domain slider to show the value on a curve for a given time; a curves velocity and acceleration; and a label to identify which curve corresponds to which parameter.

Displaying the Domain Slider

To toggle the display of the **Domain Slider**:

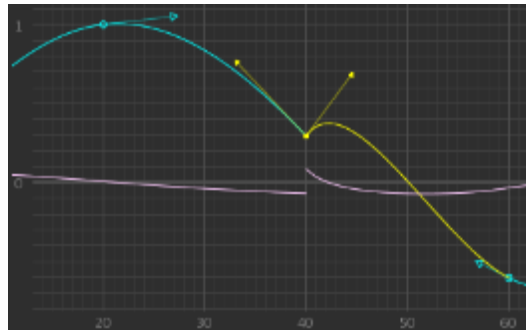
Right-click and select **Show > Domain Slider** (or press **D**).



The **Domain Slider**, the orange vertical bar, can be moved left and right across the frame range to display the value for the highlighted curve at a particular frame.

Displaying a Velocity Curve

You can use a velocity curve for a parameter to help you spot non-tangential keyframes; these are characterized by breaks in the velocity curve. Non-tangential keyframes can be jarring when making realistic movement through animation. The velocity curve is calculated by analyzing the changes in the y axis of the curve at small increments along the x axis.

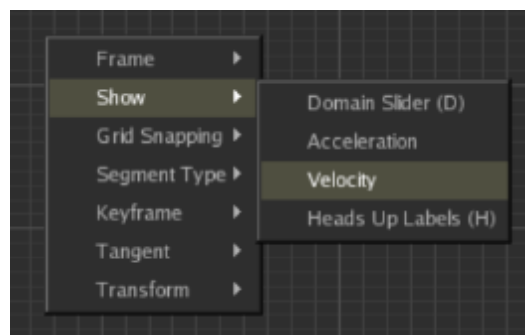


The purple velocity curve is broken (not continuous) at frame 40, as is the highlighted tangent.

To toggle the display of a curve's velocity:

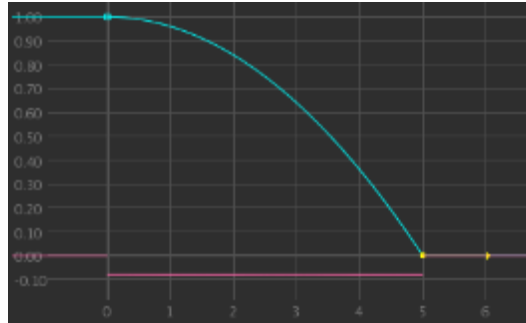
1. Select the curve(s) within the hierarchical view.
2. Right-click an empty part of the graph and select **Show > Velocity**.

The velocity curve is shown in lavender.



Displaying an Acceleration Curve

You can use the acceleration of a curve to provide a useful insight into the forces that act on that curve. For instance, an object whose only force is gravity should have a horizontal acceleration curve (assuming it doesn't hit anything).

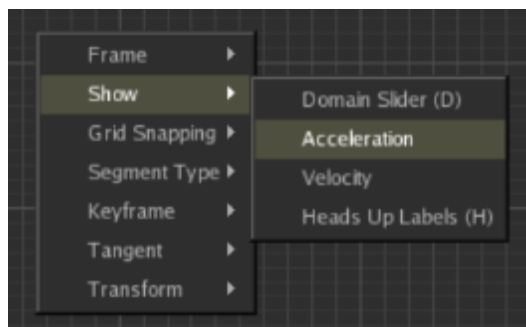


Between frames 0 and 5 the acceleration curve shows a consistent force is acting on the parameter (the acceleration curve is straight).

To toggle the display of a curve's acceleration:

1. Select the curve(s) within the hierarchical view.
2. Right-click anywhere on the graph and select **Show > Acceleration**.

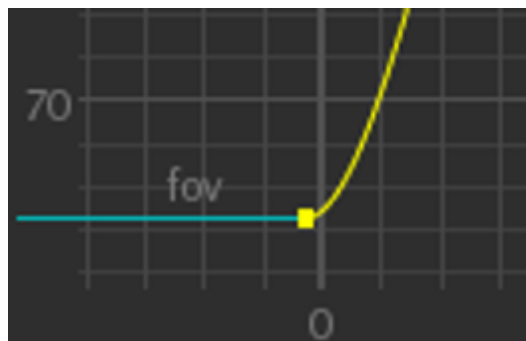
The acceleration curve is shown in pink.



Displaying Curve Labels

To toggle the display of curve labels:

Right-click and select **Show > Heads Up Labels** (or press **H**).



The curve label, based on the parameter name, sits just above the curve on the left-hand side.

Snapping Keyframes

When moving keyframes within the **Curve Editor** tab, you can snap their values in place. Snapping to the x axis affects the frame number and snapping to the y axis affects the parameter's value.

Snapping a Keyframe's Frame Number While Moving it Within the Curve Editor Tab

You can snap the frame number of a keyframe in two ways:

- Right-click and select **Grid Snapping > X Snap to Integers**.
Katana snaps keyframe changes to whole frame numbers.
- Right-click and select **Grid Snapping > X Snap to Grid**.
Katana snaps keyframe changes to the vertical grid lines.



NOTE: Selecting either of these menu options does not change the current y axis snap settings.

Snapping a Keyframe's Value While Moving it Within the Curve Editor Tab

Right-click and select **Grid Snapping > Y Snap to Grid**.

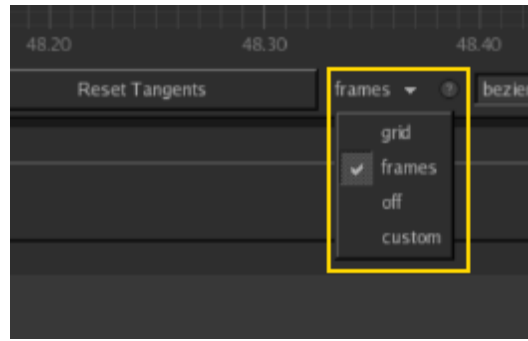
Katana snaps value changes to the horizontal grid lines.

Turning Off Keyframe Snapping

- Right-click and select **Grid Snapping > X Snapping Off**.
Katana no longer snaps keyframe changes in the x axis.
- Right-click and select **Grid Snapping > Y Snapping Off**.
Katana no longer snaps keyframe changes in the y axis.
- Select **off** from the dropdown menu to the right of the **Reset Tangents** button at the bottom of the **Curve Editor**.
Katana no longer snaps keyframe changes in any direction.

Katana also comes with some predefined snapping options in a dropdown menu to the right of the **Reset Tangents** button at the bottom of the **Curve Editor**. These are:

- **off** - Katana no longer snaps keyframe changes in any direction.
- **frames** - Katana snaps the x axis to whole frame numbers but does not snap the keys in the y axis.
- **grid** - Katana snaps the keyframes to grid intersection points.
- **custom** - The last snap setting you selected that does not match **frames**, **grid**, or **off**. (This option only becomes available once you have made a snap setting change that does not match frames, grid, or off.)



Cycling Through the Preset Snapping Options (Off, Frames, and Grid)

Right-click and select **Grid Snapping** > **Cycle Snapping** (or press **S**).

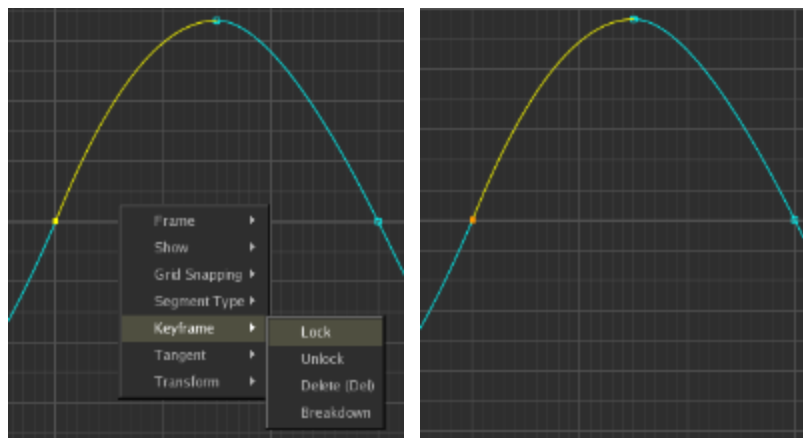
Locking, Unlocking, and Deleting Keyframes

Locking Keyframes to Prevent Accidental Editing

1. Select the keyframes to lock.
2. Right-click and select **Keyframe** > **Lock**.
Katana locks the keyframes and turns them orange.



NOTE: Locking a keyframe only applies to inside the **Curve Editor** tab.



Unlocking Keyframes

1. Select the keyframes to unlock.

2. Right-click and select **Keyframe > Unlock**.

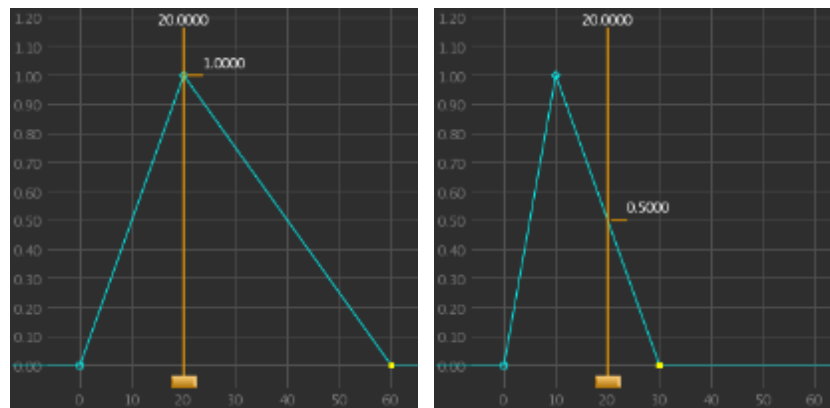
Katana unlocks the keyframes and turns them yellow.

Deleting Keyframes

1. Select the keyframes to delete.
2. Right-click and select **Keyframe > Delete** (or press **Delete**).

Turning a Keyframe into a Breakdown

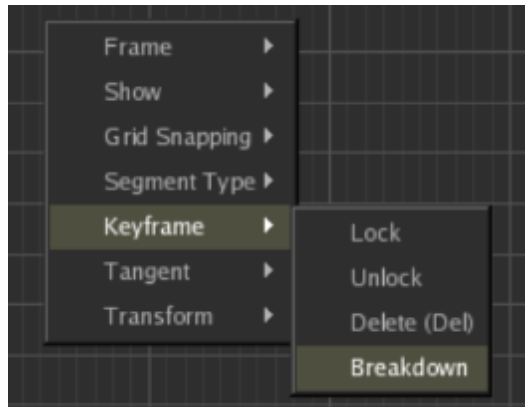
Katana supports a special kind of keyframe known as a breakdown. Breakdowns help you describe the motion between two keyframes by providing an intermediate value. Breakdowns maintain the same relative time with the keyframes either side, this helps maintain timing. For instance, with keyframes on frames 0 and 60 and a breakdown on frame 20, moving the keyframe on frame 60 to frame 30 would automatically move the breakdown to frame 10, thereby maintaining the 1:2 ratio of frames before and after. If a breakdown falls at the beginning or end of a curve, then moving the keyframe next to it moves the breakdown.



When the keyframe on frame 60 is moved to frame 30, the breakdown on frame 20 automatically moves to frame 10.

To convert a keyframe into a breakdown:

1. Select the keyframe(s) to convert.
2. Right-click and select **Keyframe > Breakdown**.



TIP: To change a breakdown back to a keyframe, repeat the steps above.



NOTE: Breakdowns are only different to keyframes while within the Curve Editor. Elsewhere, such as within the Dope Sheet, breakdowns are treated as normal keyframes.

Changing a Segment Function

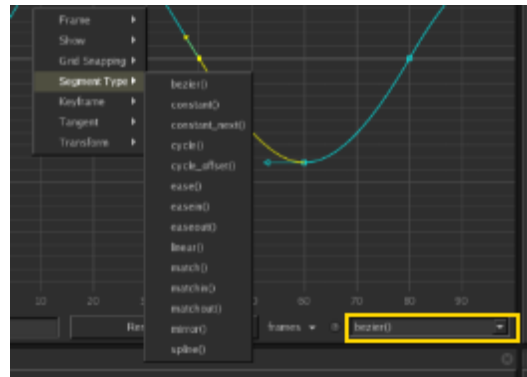
Katana interpolates the values between one keyframe and the next based on the segment function assigned to the first of the two keyframes. Three special segment functions can also be assigned to the segment before the first keyframe or after the last: `cycle()`, `cycle_offset()`, and `mirror()`.

Changing the Segment Function for Either a Keyframe or for the Segment at the Beginning or End of a Curve

1. Select the keyframe(s) or segment to change (to select a segment click on it).
2. Then, either:
 - Right-click and select **Segment Type** >

OR

- Select the segment function from the dropdown menu in the bottom right corner of the **Curve Editor**.

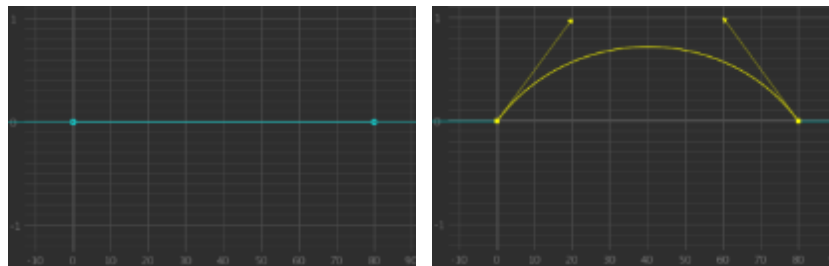


Available Segment Functions

The following are a list of available segment functions:

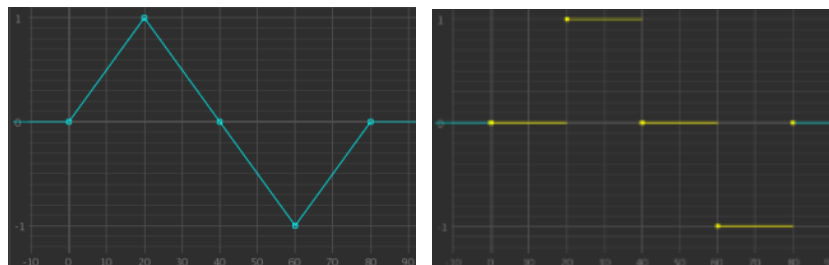
- **bezier()**

The bezier segment function is the most versatile. It uses four points - the keyframes at the start and end, and two control points - to define the segment. The control point position is shown with an arrowhead. The weight of a control point, which determines how strong its influence is over the generated curve, is determined by the length of the handle.



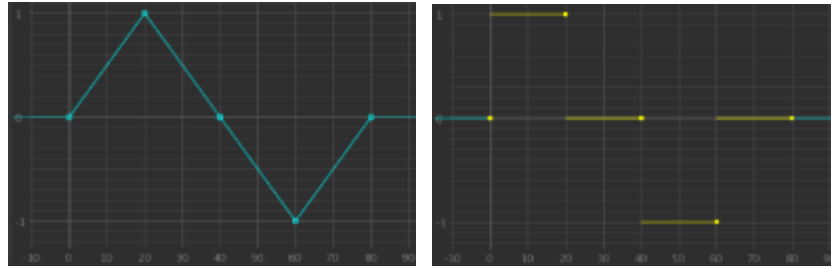
- **constant()**

The constant segment function uses the keyframe's value for the entire segment.



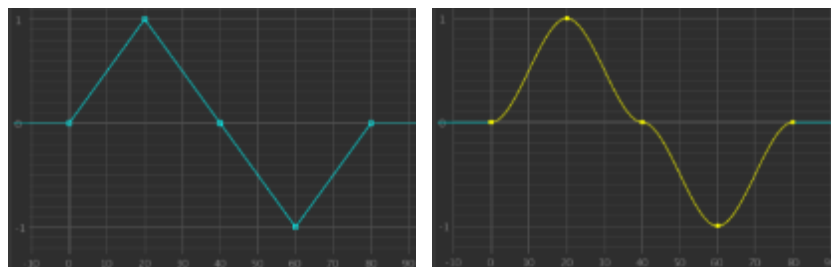
- **constant_next()**

The `constant_next` segment function uses the next keyframe's value for the entire segment.



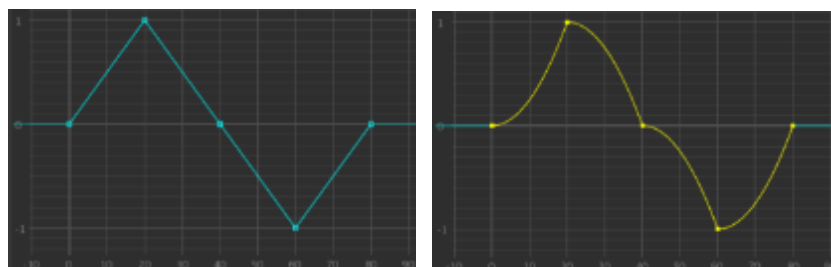
- **ease()**

The ease segment function flattens out the segment at its beginning and end. This is similar to having flat tangents on the two control points when using bezier curves.



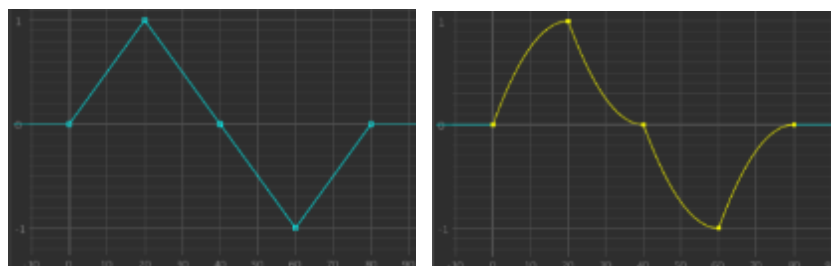
- **easein()**

The easein segment function starts the segment flat and then maintains the same acceleration until it reaches the next keyframe. This results in the velocity curve for the segment being a straight line that starts at zero.



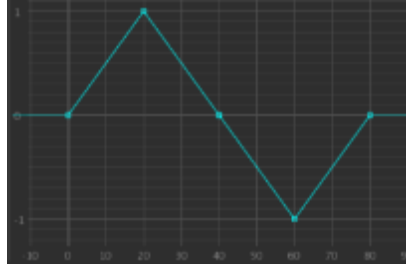
- **easeout()**

The easeout segment function finishes the segment flat while maintaining a constant acceleration throughout the segment. This results in both the velocity curve for the segment being a straight line that ends at zero.



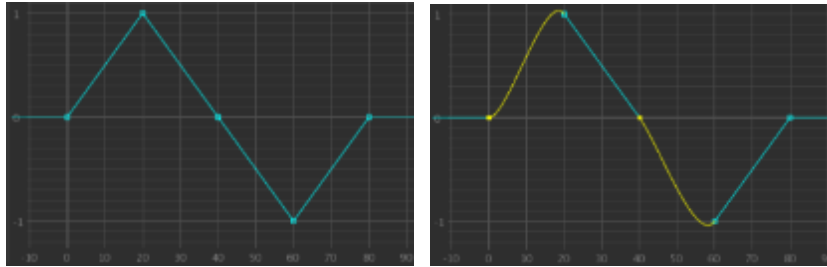
- **linear()**

The default segment function. The values from one keyframe move in a straight line to the next keyframe.



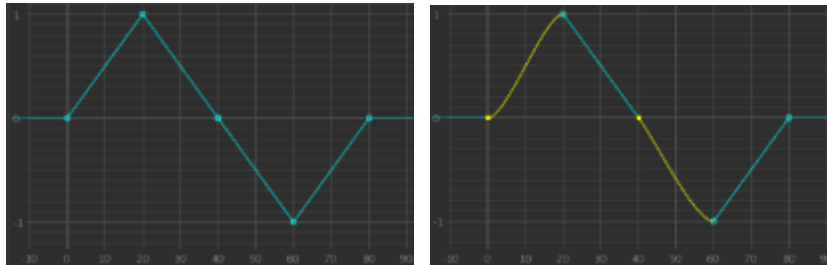
- **match()**

The match segment function gives the segment the same velocity (rate of change) at both the start and end of the segment.



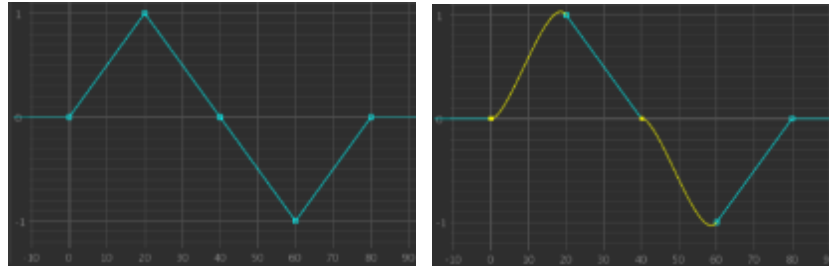
- **matchin()**

A segment with the matchin segment function begins with a velocity that matches that at the end of the previous segment, the segment ends with zero velocity. This has the effect of making the tangent at the start match the slope of the previous segment and the tangent at the end flat.



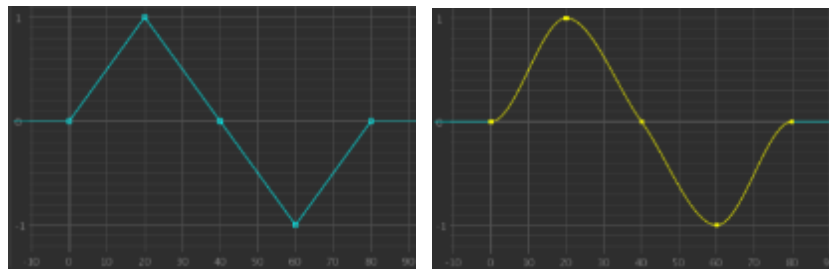
- **matchout()**

A segment with the matchout segment function begins with zero velocity and ends with a velocity that matches that at the beginning of the next segment. This has the effect of making the tangent at the start flat and the tangent at the end match the slope of the next segment.



- **spline()**

The spline segment function uses the Catmull-Rom spline function that uses four keyframes to calculate the value at a given frame. As the frame approaches a keyframe, the curve tends towards the value at the keyframe, eventually passing through it.

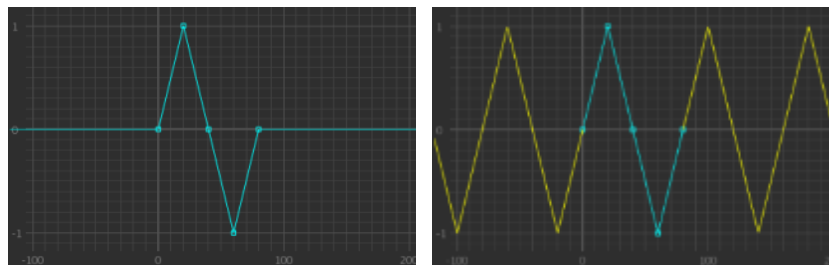


Available Extrapolation Functions

Extrapolations functions are used to extend the behavior of a curve before the first keyframe and after the final keyframe. The available options are:

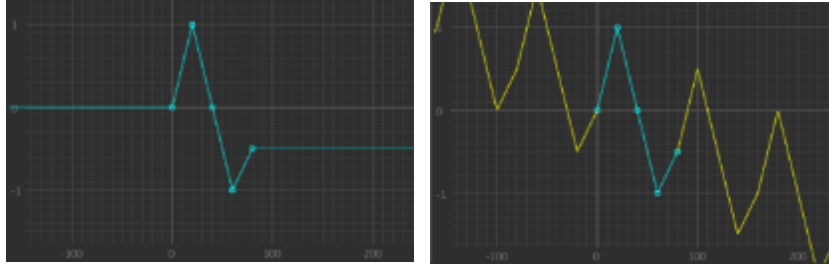
- **cycle()**

The cycle extrapolation function repeats the curve an infinite number of times either before (if applied to the segment before the first keyframe) or after (if applied to the segment after the last keyframe).



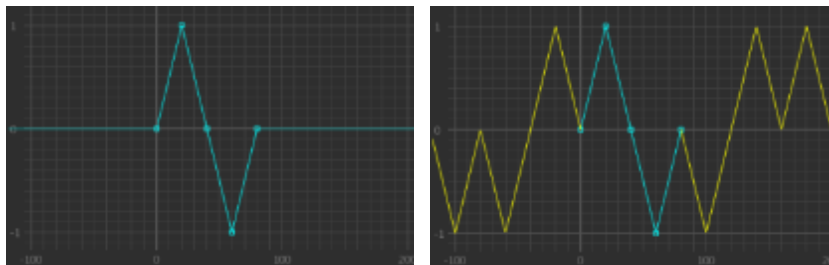
- **cycle_offset()**

The cycle_offset segment function only works on the segments at the start or end of a curve. It should not be used on a keyframe. It repeats the curve an infinite number of times; each time the curve repeats the new beginning keyframe starts from the end keyframe from the previous cycle, thus offsetting the curve.



- **mirror()**

The mirror segment function only works on the segments at the start and end of a curve. It continuously flips the curve vertically.

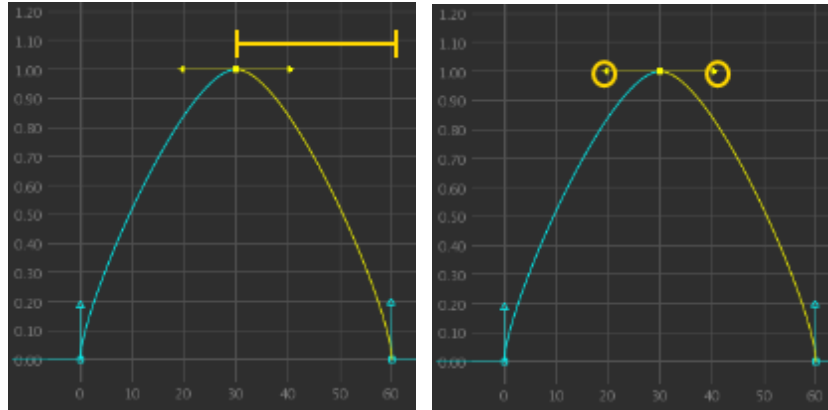


TIP: It is also possible for you to type your own segment or extrapolation function in the dropdown menu. The function must use Python syntax: **x()** can be used to represent the current frame. For instance, $\sin(x() \cdot \pi / 20)$.

Changing the Control Points of a Bezier Segment Function

Of all the segment functions, the bezier is the most versatile. With the addition of two control points, you have much finer control over how the curve flows between keyframes.

When you change the segment function at a keyframe, you change how the curve is interpolated from that keyframe to the next. When you change the tangent at a keyframe, you affect the control points that sit either side of that keyframe.



The range of any changes to the segment function.

The control points influenced by tangent changes.

Changing the Tangent Type at a Keyframe

1. Select the keyframe(s) to change the control points.
2. Right-click and select **Tangent > Type > ...**.

Changing Between Weighted and Non-Weighted Tangents

1. Select the keyframes whose tangents you want to change.
2. Right-click and select **Tangent > Weighted**.
Katana toggles the tangent between weighted and non-weighted.

With a non-weighted tangent using the manipulator only changes the angle of the control point. Weighted tangents enable you to change the amount of influence a control point has over the segment function by changing the distance from the keyframe to the end of the tangent. The bigger the distance, the more influence the control point has.

Available Tangent Types

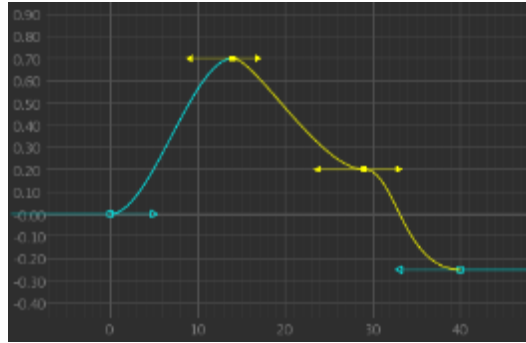
The following are a list of tangent types:

- **Fixed**

The Fixed tangent type doesn't change the current control points but they no longer update as keyframes around them are moved. This becomes the tangent type once any tangent has been manually moved.

- **Flat**

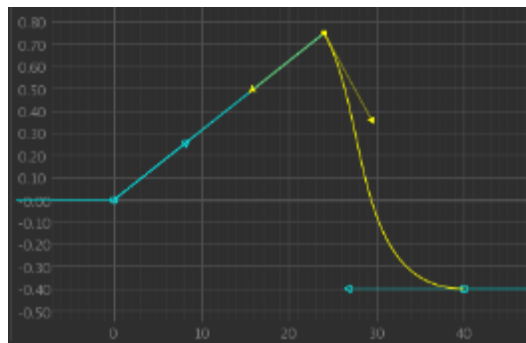
The Flat tangent type makes the control points sit horizontally either side of the keyframe.



All the keyframes are using the Flat tangent type.

• Linear

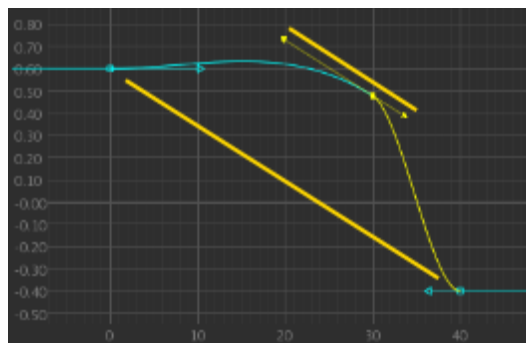
The Linear tangent type places the control point directly in line with the keyframe that acts as the other anchor point for the segment. If both control points for a bezier segment are linear, the segment is a straight line from one keyframe to the next.



The first and middle keyframes use the linear tangent, the right keyframe does not.

• Smooth

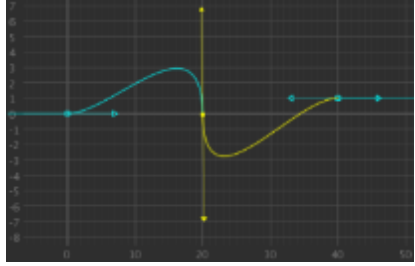
The Smooth tangent type places the control points either side of a keyframe forming a line that runs parallel to a line formed by the keyframes either side.



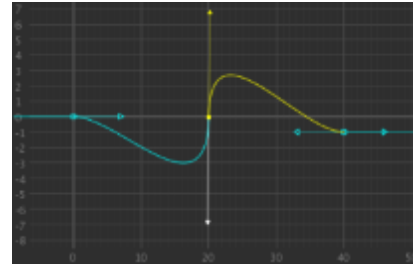
The line formed by the control points remains parallel to the line created by the keyframes.

• Smooth Normal

The Smooth Normal tangent type places the two control points vertically in line with the keyframe. Whichever keyframe is higher between the keyframes to the left and right, controls the direction of the curve. Should the keyframes to the left and right be equal, both control points are placed vertically below.



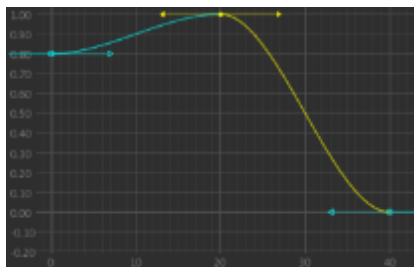
With the right keyframe above the left, the curve goes down through the middle keyframe.



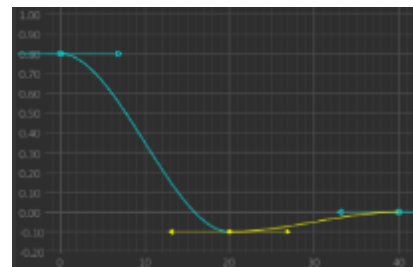
With the right keyframe below the left, the curve goes up through the middle keyframe.

• Plateau

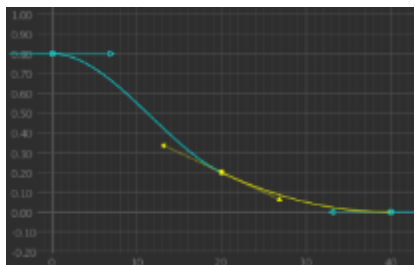
The Plateau tangent type uses the Flat and Smooth tangent types depending on its keyframes location relative to the keyframes on either side. If the keyframes on either side are both above or both below the tangent's keyframe, then the Flat tangent type is used. If the tangent's keyframe falls between the values for the keyframes on either side, then the Smooth tangent type is used. When using the Smooth tangent type, if one of the control points for the tangent would fall outside the range between the keyframes on either side, then that control point converts to the Flat tangent type instead.



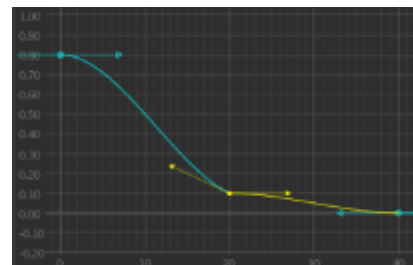
Here the Plateau tangent type uses the same algorithm as the Flat tangent type.



Once again the Flat tangent type is used.



Here the Plateau tangent type uses the same algorithm as the Smooth tangent type.



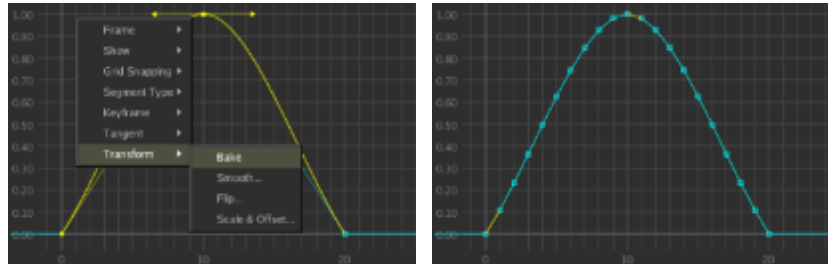
As the lower control point would drop below the keyframe to the right, that control point becomes Flat.

Baking a Segment of the Curve

Baking a segment of the curve converts the interpolated values at each frame of the segment into keyframes.

To bake a segment of the curve:

1. Select the keyframe at the start of the segment.
2. Right-click and select **Transform > Bake**.



TIP: Multiple segments can be baked at once by selecting multiple keyframes.

Smoothing a Segment of the Curve

Smoothing a segment of the curve makes the curve flatter - reducing its peaks and troughs.

To smooth a segment of the curve:

1. Select the keyframe at the start of the segment you want to smooth.
2. Right-click and select **Transform > Smooth...**

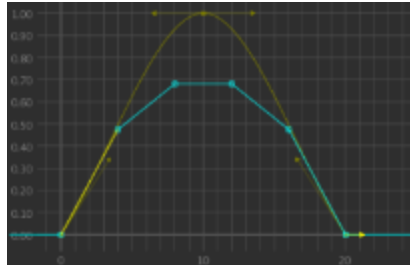
The **Smooth** dialog displays.



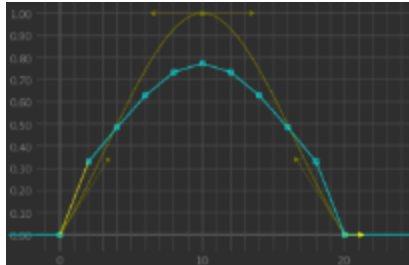
3. Change the values within the dialog where appropriate:
 - **Step Size** - how often to create a keyframe.
 - **Radius** - how much to smooth the curve (higher values for smoother, lower values for closer to original).
 - **Filter** - which algorithm to use, **Triangle** or **Box**.
4. Click **Apply** to smooth the curve.



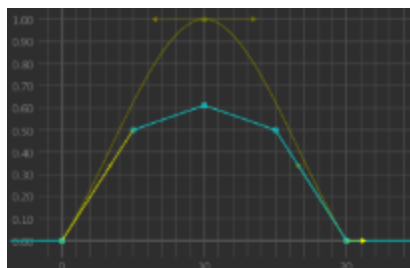
NOTE: For the best results, smooth multiple segments at once by selecting a number of keyframes together.



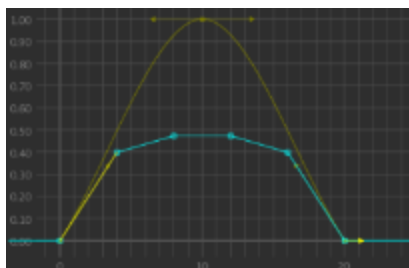
Smoothing with the default settings: **Step Size 4**, **Radius 2**, and **Triangle Filter** (the original curve is ghosted out).



Smoothing with **Step Size 2** and **Radius 4**.



Step Size 5 and **Radius 2**.



Step Size 4 and **Radius 4**.

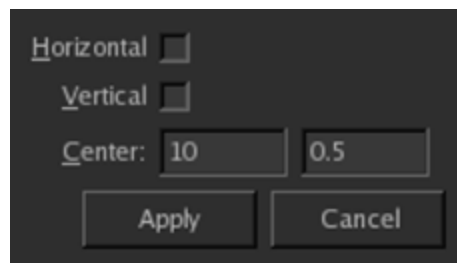
Flipping the Curve Horizontally or Vertically

You can flip a curve either horizontally or vertically.

To flip a curve horizontally or vertically:

1. Select a curve or a curve's keyframe.
2. Right-click and select **Transform > Flip...**

The **Flip** dialog displays.



3. Select whether you want to flip the curve horizontally or vertically or both:
 - **Horizontal** (press **Alt+H**) - flips the curve horizontally.
 - **Vertical** (press **Alt+V**) - flips the curve vertically.
 - **Center** (press **Alt+C**) - the point at which to flip the curve (if a keyframe is selected it defaults to that keyframe position).
4. Click **Apply** to flip the curve.

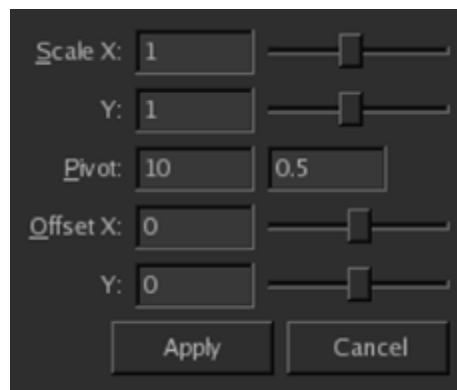
Scaling and Offsetting a Curve

Katana gives you the ability to scale or offset a curve.

To scale or offset a curve:

1. Select the curve or a curve's keyframe.
2. Right-click and select **Transform > Scale & Offset...**

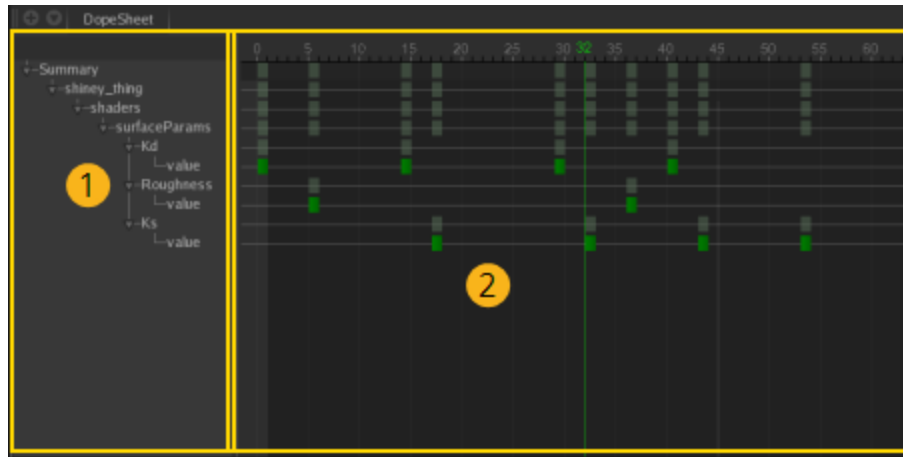
The **Scale & Offset** dialog displays.



3. Change the values within the dialog to get the desired effect:
 - **Scale** (press **Alt+S**) - scales in either the x direction (changing timing) or y direction (changing the parameter value). Negative values reflect the curve about the values entered in the **Pivot** fields.
 - **Pivot** (press **Alt+P**) - the point about which to scale (if a keyframe is selected it defaults to that keyframe position).
 - **Offset** (press **Alt+O**) - moves the curve in the direction of the offset.
4. Click **Apply** to effect the curve.

Dope Sheet Overview

In the Dope Sheet, you can manipulate keyframes by either retiming (sliding them left or right) or copy and pasting. The Dope Sheet's simple interface makes it easy for you to see keyframe timings across multiple parameters, whereas the Curve Editor can become cluttered when dealing with more than one curve.



1. The Dope Sheet has a hierarchical view down the left-hand side.
2. The main area has time (in frames) across the top and blocks (to signify keyframes) at the intersection of their parameter on the left-hand side and their frame number above.

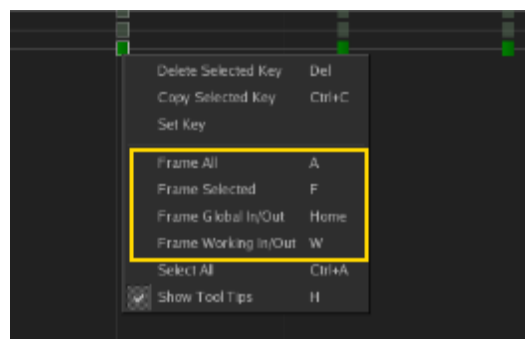


NOTE: Within the Dope Sheet breakdowns are treated as normal keyframes - this means they do not move automatically when the keyframes either side are moved.

Changing the Displayed Frame Range

There are multiple ways for you to change the frame range displayed within the Dope Sheet. You can:

- Scroll the mouse-wheel; to zoom in scroll up and to zoom out scroll down.
- **Alt**+middle-click and drag.
- Press **+** (plus key) to zoom in or **-** (minus key) to zoom out.
- Right-click and select **Frame All** (or press **A**) to have the frame range zoom to fit all the keyframes.
- Right-click and select **Frame Selected** (or press **F**) to have the frame range zoom to fit only the selected keyframes.
- Right-click and select **Frame Global In/Out** (or press **Home**) to have the frame range go from the project settings' **inTime** to the project settings' **outTime**.
- Right-click and select **Frame Working In/Out** (or press **W**) to have the frame range go from **In** to **Out** from the **Timeline**.



Panning the Displayed Frame Range

To pan the displayed frame range within the Dope Sheet, middle-click and drag.

Selecting Keyframes

The Dope Sheet has standard controls for selecting single or multiple keyframes.

Selecting a Keyframe

- click on it,

OR

- drag a marquee around it.

Selecting Multiple Keyframes

- drag a marquee around all the keyframes you want to select,

OR

- right-click and select **Select All** (or press **Ctrl+A**) to select all visible keyframes.

Adding to a Selection

Click or drag a marquee over the keyframe(s) while holding the **Shift** key.

Removing from a Selection

Click or drag a marquee over the keyframe(s) while holding the **Ctrl** key.

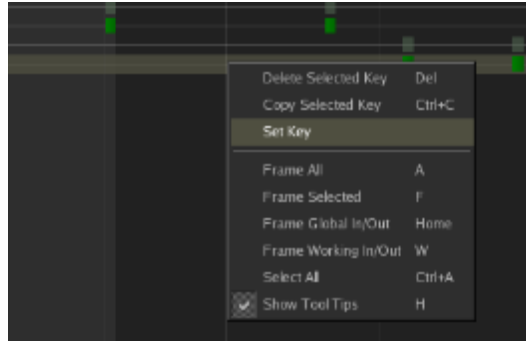
Moving Keyframes

To move keyframe(s):

1. Select the keyframe(s) to move.
2. Click on one of the selected keyframe(s) and drag left or right.

Creating a Keyframe from an Interpolated Value

At times you may want to convert an interpolated value into a keyframe; you can achieve this by right-clicking at the intersection of the frame and parameter (this is where the keyframe block displays) and selecting **Set Key**. A new keyframe is created with the same value as previously interpolated at that frame.

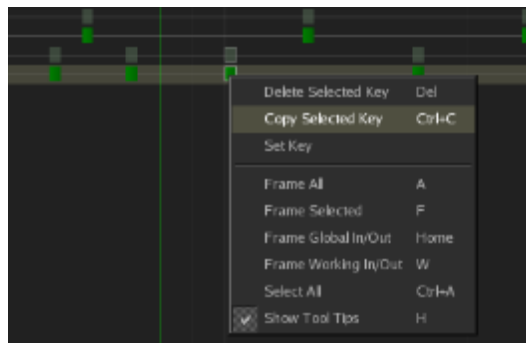


Copy and Pasting Keyframes

The **Dope Sheet** provides the simplest method for copying and pasting keyframes.

Copying Keyframe(s)

1. Select the keyframe(s) to copy.
2. Right-click and select **Copy Selected Key(s)** (or press **Ctrl+C**).



Pasting Keyframe(s)

Right-click and select **Paste Key**.

If you right-click on an empty part of the **Dope Sheet**, the keyframe(s) are inserted in the same parameter from which it was copied at the point shown by a ghosted vertical line.

If you right-click horizontally in line with a parameter, the keyframe(s) are added there. The precise positions are highlighted when you first right-click.

OR

1. Using the **Timeline**, move the current frame to where you want to insert the new keyframe(s).
2. Press **Ctrl+V**.

Deleting Keyframes

To delete keyframe(s):

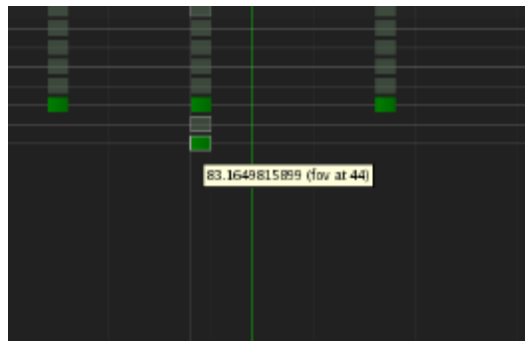
1. Select the keyframe(s) to delete.
2. Right-click and select **Delete Selected Key(s)** (or press **Del**).



TIP: When creating, copying, or deleting keyframes within the Dope Sheet, it is a good idea to keep checking the new curve within the Curve Editor to make sure the curve segments are interpolated using the right segment function.

Toggling Tooltip Display

To toggle tooltip display, right-click and select **Show Tool Tips** (or press **H**). The value, parameter name, and frame number for the keyframe display.



Groups, Macros, & Super Tools

Groups, Macros and Super Tools are ways of creating higher-level compound nodes out of other nodes. Groups and macros are created in the Katana UI, while Super Tools are created using Python.

Groups

Groups are simply nodes that group together a number of other nodes into a single one. They are typically used to simplify the Node Graph by collecting nodes together. The simplest way to create a group is by selecting a number of nodes in the Node Graph and pressing **Ctrl+G**. A new Group node is created, with the previously selected nodes as it's children. Any connections between selected nodes are preserved. You can also create an empty group by choosing **Group** from the **Tab** shortcut **node creation** menu. Group nodes can also be duplicated like any other node, creating duplicates of any child nodes.

As well as standard Group nodes - which can have child nodes of any node type - there are two special convenience Group nodes to make it easier to collect multiple nodes of the same type: GroupStack and GroupMerge. GroupStacks are for nodes with a single input and output (such as MaterialAssign), and connect all the internal nodes together in series. GroupMerge nodes are for nodes with a single output that don't require any input, and connect all the internal nodes in parallel using a Merge node. To create a GroupStack or GroupMerge node of a particular node type, simply create a GroupStack or GroupMerge and **Shift**+middle-drag a node of the desired type into it, from the **Node Graph** tab.

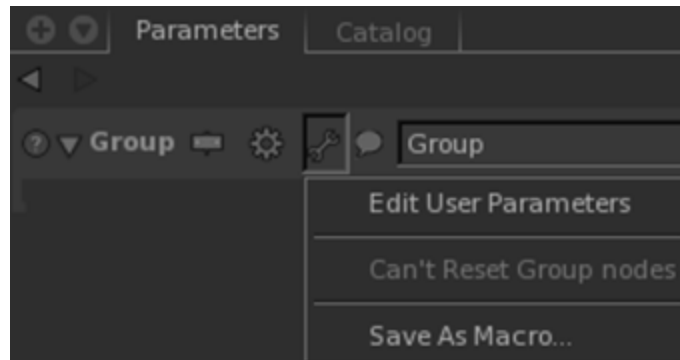
Group nodes - as with any other node - can have user parameters, with which you can add parameters to a node, and interactive connections between parameters. See the [Adding User Parameters](#) section for more on adding and using user parameters.

A Group node comprised of nodes with no incoming or outgoing connections outside the group, has no input or output ports itself. For a group to have input or output ports, its child nodes must have suitable connections outside the group, or they must be explicitly created. For more on Group nodes, and their incoming and outgoing connections, see the *Group Nodes* section in the *Katana Technical Guide*.

Macros

Macros are nodes that wrap any single node, or group of nodes, and publish them so that their state is saved and they can be recalled in other projects. You can create a macro from any node by using the **wrench** menu at the top of the node's **Parameters** tab, and choosing **Save As Macro...** although Macros are more typically - and more

usefully - made from groups. You save a macro from a Group node in the same way you would from any other node (by choosing **Save As Macro...** from the **wrench** menu in a group's **Parameters** tab).



Once you have created a macro, it can be added to a project as you would add a regular node, including from the **Tab node creation** menu. By default - if no other directory options are set - macros are saved into the **Home** directory in **.katana/Macros/_User** and are given the suffix **_User**.

You can also read macros from other directories included in your `$KATANA_RESOURCES` environment variable. Macros are picked up from sub-directories called **Macros** and take the name of the parent directory as suffix. For example, if you point your `$KATANA_RESOURCES` to **/production/katana_stuff/_studio** you can put macros in **/production/katana_stuff/studio/Macros** and they can automatically be suffixed with **_studio**.

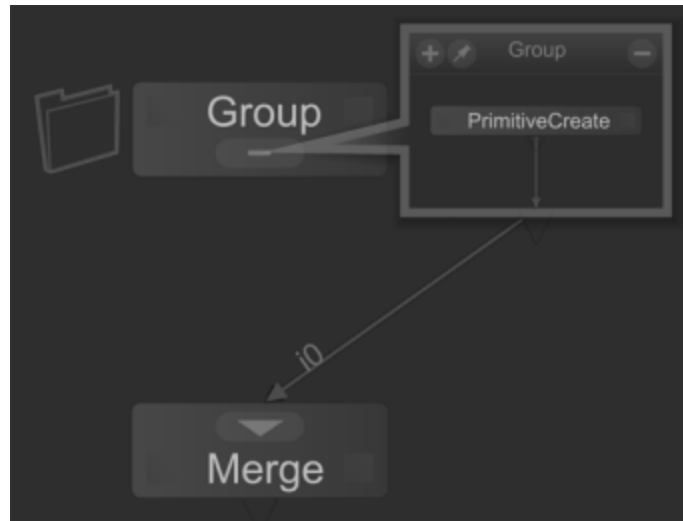
If you want the macro to have input or output ports you need to make sure that there are connections to other external nodes when you create the group. For each original connection from an internal node to an external one, the macro creates an appropriate port.

Adding User Parameters

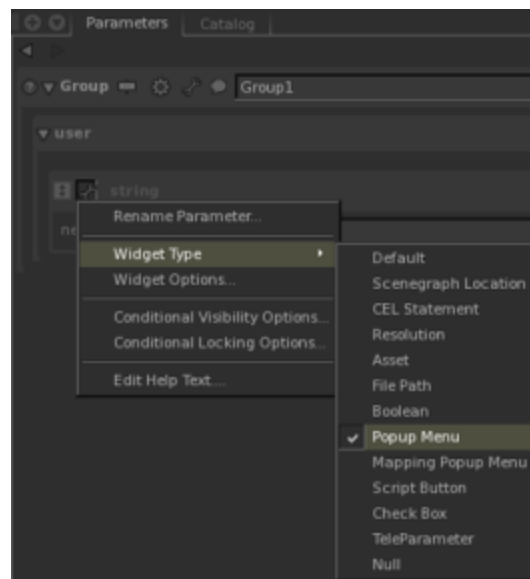
You can add user parameters to any node, but they're particularly useful in groups and macros, where user parameters on the parent node can drive parameters on child nodes, or even on nodes in the recipe outside of the group or macro.

An example for adding user parameters is as follows:

1. Start with a recipe, as shown below, consisting of a Group node. Add a PrimitiveCreate node, and a connection out of the group from the PrimitiveCreate to a Merge node.



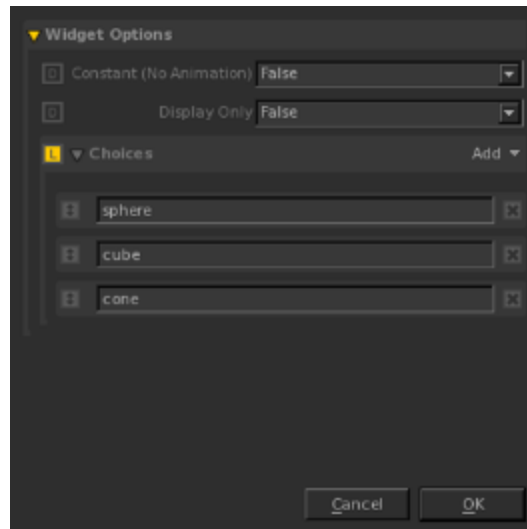
2. Select the Group node and press **Alt+E** to edit it.
3. In the Group node's **Parameters** tab, choose **Edit User Parameters** from the **wrench** menu.
4. Choose **Add > String**.
A new user parameter of type string is created.
5. Change the widget type of your user parameter to **Popup Menu**.
In the **wrench** menu of the new parameter, choose **Widget Type > Popup Menu**.



This changes the user parameters widget type to pop-up menu, where each entry in the menu is a string. For more on the types of user parameters available, see [User Parameters and Widget Types](#) on page 281.

6. Edit the pop-up menu to add new entries, each corresponding to a valid PrimitiveCreate node type.
In the **wrench** menu of the new parameter, choose **wrench > Widget Options...**
In the resulting Widget Options Window, choose **Add > New Entry**, and add new entries that read **sphere**, **cube**, and **cone**.

7. Link the user parameter pop-up menu to the **type** parameter of the PrimitiveCreate node.
Right-click on the user parameter pop-up menu and choose **copy**, then select the PrimitiveCreate node. Right-click on its **type** parameter, and select **Paste > Paste Expression**. The **type** parameter's background turns blue to let you know it's set by an expression.
8. Finish editing the Group node's user parameters by choosing **Finish Editing User Parameters** from the Group node's **wrench** menu.
9. Select the group, and change the user parameter pop-up menu between **sphere**, **cube**, and **cone**. The **type** parameter of the PrimitiveCreate node changes to match.



TIP: In the example above, you manually created a pop-up menu and populated it with suitable entries. Katana offers a shortcut to automatically create and populate user parameters. Start at step 3 from the example above, then **Shift**+select the PrimitiveCreate node as well as the group. **Shift**+middle-drag from the PrimitiveCreate node's **type** parameter onto the Group nodes **Add** menu. A new user parameter of the correct type (in this case a pop-up menu), populated with the applicable entries is created.

Note that you still need to link the new menu to the PrimitiveCreate node's **type** parameter, as described in step 7 above.

Conditional Behavior

You can make the behavior of user parameters within a Macro, or Group conditional, dependent on any user parameter values. For example, create a group, containing a scene that includes PRMan and Arnold shaders. Select which shader to use from a pop-up menu, and show and hide shader options using conditional behavior.

Conditional Visibility Example

1. Create a Katana scene with a PrimitiveCreate node, a Material node, a CameraCreate node, and a Merge node. Connect the outputs of the PrimitiveCreate, Material, and CameraCreate nodes to inputs on the Merge node.
2. Create a MaterialAssign node, and place it downstream of the Merge node, then add GafferThree and RenderSettings nodes in series, downstream of that. Finally, add a Render node.



NOTE: You can overload Material and GafferThree nodes with more than one shader type. For example, a Material node can hold both Arnold and PRMan shaders. At render time, only shaders relevant to the selected renderer are considered.

3. In the Material node, add a PRMan surface shader of type **KatanaPhong**, and an Arnold surface shader of type **standard**.
4. In the GafferThree node, add both a PRMan **spotlight**, and an Arnold **spotlight**, switching profiles to do so. Add two lights and, in their respective **Material** tabs, choose **Add Shader**, then choose either an Arnold or PRMan light. From the dropdown menus, choose **spot_light** for the **arnoldLightShader**, and **KatanaSpotlight** for the **prmanLightShader**. (You can also use other renderers and their shaders if you have access to them instead PRMan and Arnold.)
Position the lights, and remember to set the **decay_type** in the **arnoldLightShader** to **constant**, to match the default decay type of a PRMan spotlight (the Arnold default is **quadratic**).
5. Select all of the nodes, bar the Render node, and enter **G** to make a Group node containing all of the selected nodes. The result is a Group node, with a single output, connected to a Render node.
To switch between PRMan and Arnold rendering, you can go into the Group node, and change the RenderSettings node **renderer** parameter, but to streamline this operation, you can add a pop-up menu to the UI of the Group node, and link by expression to the renderer parameter on the RenderSettings node.
6. Select the Group, and select the  > **Edit User Parameters**.
7. Select **Add > String**, and then the new parameter's  > **Widget Type > Popup Menu**.
8. Select the new parameter's  > **Widget Options...**
9. In the widget options window, select **Add > New Entry**, so there are two entries in the menu. Edit one entry to read **arnold** and the other to read **prman** then click **OK**.
10. In the Group node's **Parameters** tab, right-click on the pop-up menu widget, and select **Copy**.
11. Expand the contents of the Group node in the **Node Graph**. Select the nodes that you want to edit and use the **Alt+E** keyboard shortcut to edit all of their parameters in the **Parameters** tab. Right-click on the node's **renderer** parameter, and select **Paste Expression**.
The background of the **renderer** parameter turns blue, to indicate that it's driven by an expression.

The value of the Render Settings node **renderer** parameter is linked by expression to the selected entry in the Group node's pop-up menu. If you select the group's  > **Finish Editing User Parameters**, the pop-up menu displays in the group's **Parameters** tab, and its value selects the renderer.

The state of the pop-up menu can also conditionally affect visibility of other user parameters in the Group node. Create new user parameters on the group to control the diffuse color values on the PRMan and Arnold shaders contained within it. Add conditional behavior to only show the color controls for shaders relevant to the selected renderer:

1. In the Group node's **Parameters** tab, select the  > **Edit User Parameters**, then **Add > Color, RGB**, twice.
2. Right-click on the first **Color, RGB** user parameter, and select **Copy**. Expand the contents of the group in the **Node Graph**, then **Shift**+select the Material node.
3. Expand the parameters for the **prmanSurfaceShader**, right-click on the **kd_color** parameter, and select **Paste Expression**.
4. In the Group node's **Parameters** tab, right-click on the second **Color, RGB** user parameter, and select **Copy**. Expand the contents of the group in the **Node Graph**, then **Shift**+select the Material node.
5. Expand the parameters for the **arnoldSurfaceShader**, right-click on the **kd_color** parameter, and select **Paste Expression**.

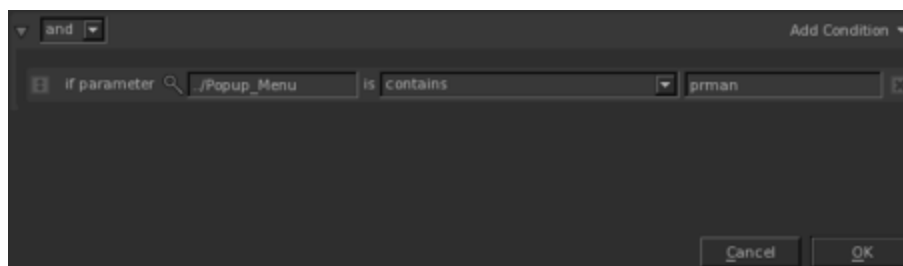
The diffuse color values of the PRMan and Arnold shaders are linked by expression to the color widgets in the groups parameters. If you select the group's  **wrench** icon > **Finish Editing User Parameters**, the color widgets display in the group's **Parameters** tab, and their values affect the diffuse colors of the PRMan and Arnold shaders.

Only one shader at a time is considered though, so it would be useful to hide the settings for the shader not applicable to the selected renderer.

1. In the Group node's **Parameters** tab, select the  > **Edit User Parameters**, Select the first **Color, RGB** widget's **wrench** menu > **Conditional Visibility Options**.

The conditional visibility options editor sets and / or conditions for showing the selected widget. Conditions are evaluated against a specified user parameter, in the format **if <selected parameter> is <selected condition> relative to an entered value, then show the widget**.

2. In the Conditional Visibility Options window, choose **Add Condition > contains**. In the text entry field, enter **PRMan**.



The condition in this case is **if** <the popup menu> **contains** < the string **PRMan**> then show the target user parameter.

3. In the **Conditional Visibility Options** window, click on the  icon to choose the user parameter to test against, and select the pop-up menu from the list.
4. Repeat the process above for the second **Color, RGB** widget, and the **Arnold** entry in the pop-up menu.

If you select the group's  > **Finish Editing User Parameters**, and view the completed Group node's **Parameters** tab, only one color widget at a time displays in the group's **Parameters** tab. Which one that is, is dependent on the value chosen in the pop-up menu.

Super Tools

Super Tools are like a Macro but written in Python. The nodes inside are created and controlled using Python scripts, and you can customize how parameters are presented using PyQt. For more information on writing Super Tools and how to use them in Katana, refer to the *Group, Macros, and Super Tools* chapter in the *Katana Technical Guide*.

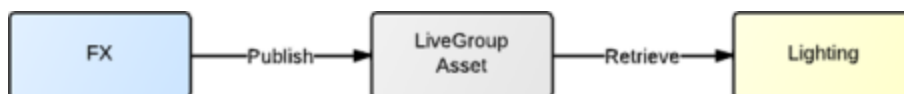
LiveGroups

A LiveGroup node provides a way for you to import another Katana project into the current project, and reload it every time the current project is updated, either automatically (for example, on scene load or before batch rendering) or manually (through context menu options). A LiveGroup node's source is expected to contain a Group or Group-like node in its root level.

When loading a LiveGroup source scene, the first loaded Group node in the source scene defines the user parameters and child node contents of the target LiveGroup. LiveGroups provide a number of useful constructs for collaborative work between users, for sharing nodes between show, sequence, and shot levels, and for users working in parallel on the same shot.

There are two primary cases for using LiveGroups: collaborative work between departments and collaborative work within a department. As an example of collaborative work between departments, an FX artist would pass a Katana project to a lighting artist that can then use that project as part of a lighting scene by loading it into a LiveGroup node.

In this example, the FX artist is only interested in publishing their setup, while the lighting artist is only interested in importing the published project, as shown in the diagram below:



As an example of collaborative work within a department, a shot or sequence lighting artist would make changes to a LiveGroup source on a shot, and could then publish the source scene back to the shot or sequence for other lighting artists to pick up, as shown in the diagram below:



In addition to the existing **.katana** file extension, the **.livegroup** file extension has been introduced. For backwards compatibility, it is still possible to use a Katana project with the **.katana** file extension as a LiveGroup source. When publishing new LiveGroup sources using the **Publish...** or **Publish and Finish Editing Contents...** menu options, the **.livegroup** file extension is used.

Any Katana project file can be used as a source of a LiveGroup node, and LiveGroup sources can be published as assets through Katana's Asset API. When a LiveGroup node's contents are exported to a file, the **.livegroup** file extension is used to create it. When a LiveGroup is imported, you can choose whether to import either **.katana** or **.livegroup** files. Files with the extension **.macro** are not listed by default but if there are macros present in the file structure, you can select the macros option in the **Types** field in the **Import Livegroup** dialog.

For more information on Asset Management and the asset publishing process, see [Asset Management](#), or refer to the chapter on the *Asset Management System Plug-in API* in the *Katana Technical Guide*.

By default, exported **.katana** files are binary, gzip-compressed archives in a **.tar** format, containing an XML scene description file as data. This format is archived and compressed because nodes may contain binary data. In contrast, the **.livegroup** extension uses uncompressed, unarchived, ASCII files in a format similar to **.katana** but with a dedicated extension for LiveGroup sources. This means that LiveGroups, by default, are written as plain text, uncompressed XML files.

While **.katana** files contain project settings in addition to the nodes of the node graph document, as they represent Katana projects, **.livegroup** files only contain the XML representation of the parameters and children of the Group node that controls the LiveGroup interface and contents.

Creating a LiveGroup

You can create a LiveGroup node from scratch or by importing a LiveGroup source or Katana project as a starting place. To create a LiveGroup node, in the **Node Graph** tab:

- Click **New > Other > LiveGroup** from the menu,
- Press **Tab** and select **LiveGroup** from the node list, or
- Right-click and select **Other > LiveGroup** from the menu.

The LiveGroup node floats with the cursor. Click inside the **Node Graph** tab to place it at that location.

As mentioned above, you can also create a LiveGroup node and choose a source to import in one step by using an option in Katana's **File** menu. When a LiveGroup source is selected for import, its contents are read and the XML document structure is parsed. The first Group node, or Group-like node, found in the root level of the document is used for defining the parameters and contents of the target LiveGroup node. Other nodes, including other Group nodes, in the root level of the document are ignored.

To import a project as a LiveGroup node:

1. Select **File > Import LiveGroup...**
The **Import LiveGroup** dialog appears.
2. Select a LiveGroup source file in the file browser (see [Using the File Browser](#)).
3. Click **Accept**.

A LiveGroup node, named after the imported file, floats with the cursor inside the **Node Graph**. Click anywhere within the **Node Graph** to place it at that location.

Editing LiveGroup Parameters

The wrench menu for a LiveGroup node that is edited in the **Parameters** tab contains not only the same generic options as other nodes, such as **Edit User Parameters**, **Reset Parameters**, and **Save as Macro**, it also has a unique set of options that change dynamically depending on whether the node is in a locked or editable state. For information on accessing or editing a node's parameters, see [Editing a Node's Parameters](#).

In the **Parameters** tab, click the wrench  icon, or right-click on the LiveGroup node. The following options appear in the wrench menu, provided the node is in a locked, non-editable state:

- **Load...** - opens the **Load LiveGroup** dialog to allow you to select a LiveGroup source for the node.
- **Reload** - reloads the source file, if you already have one specified in the **source** parameter.
- **Edit Contents** - changes the state of the node to be editable, so that contents of the node can be modified. In this state, the contents of the node are no longer loaded from its source, if a source is set.
- **Convert to Group** - converts the LiveGroup node to a Group node. Note that the Group node does not retain the LiveGroup node's source parameter.



NOTE: Converting a LiveGroup node to a Group node discards any history or knowledge of the source file.

When the LiveGroup node is in an unlocked, editable state, the following menu options are available:

- **Revert** - reloads the LiveGroup from its current **source**, thus discarding any changes made while it was unlocked. The **Revert** option is only available when a **source** has been set.
- **Publish...** - opens the **Publish LiveGroup** dialog for publishing the parameters and contents of the LiveGroup node as a LiveGroup source file or asset.
- **Publish and Finish Editing Contents...** - opens the **Publish and Finish Editing Contents LiveGroup** dialog for publishing the parameters and contents of the LiveGroup node as a LiveGroup source file or asset, and for loading the published source file or asset. This locks the LiveGroup node's contents, making them non-editable.

To access any of the above menu options, right-click on the LiveGroup node, or click the wrench  icon in the **Parameters** tab. The menus found using either method display the context-sensitive **LiveGroup** menu options.

In addition, parameters can be opened in a floating pane as well as in the **Parameters** tab displayed in your preferred layout by right-clicking on the LiveGroup node and selecting **Show Parameters in Floating Pane** from the context menu.

Loading and Reloading a LiveGroup

Loading a LiveGroup source into a LiveGroup node opens the selected source file or asset, parses its contents, and uses the first Group node found in the source for the parameters and contents of the LiveGroup node. Once a

LiveGroup source has been loaded into a LiveGroup node, the source can be reloaded to pick up any changes that have been made to the file or asset outside of the current Katana session.

To load the LiveGroup node source:

1. Right-click the LiveGroup node, or click the wrench  icon in the **b** tab, and select **Load** from the menu.
The **Load LiveGroup** dialog appears.
2. Select the file or asset from the file browser to use as the **source** of the LiveGroup node.
3. Click **Accept**.



NOTE: If the LiveGroup is editable and its contents have been modified, you must confirm whether you want to proceed. Continuing to load from the source without publishing changes first results in the unsaved changes being lost. To keep any changes, refer to [Publishing a LiveGroup](#) for more information.

To reload the LiveGroup node source:

1. Right-click the LiveGroup node, or click the wrench  icon in the **Parameters** tab, and select **Reload** from the menu.
The LiveGroup node's contents are replaced with the contents of the first Group node loaded from the selected LiveGroup source.



NOTE: If the **source** parameter specifies the latest version of a LiveGroup asset, the plug-in is used to query the asset management system for the latest version of the LiveGroup asset, which is then used as the actual source file to load.

Editing the Contents of a LiveGroup

LiveGroup nodes, which are created in a locked state, can be made editable so that their contents can be freely manipulated. Once a LiveGroup node is in an editable state, it can be published as a LiveGroup source file or as a new version of a LiveGroup source asset.

When loading a LiveGroup source file into an editable LiveGroup node, the contents of the LiveGroup node become locked and non-editable. Its contents are automatically updated from the selected LiveGroup source. When a non-editable LiveGroup node is made editable, its contents are no longer automatically loaded from its selected source.

Making a LiveGroup Node Editable

Making a LiveGroup node editable allows the contents to be edited. To make a LiveGroup editable, follow the instructions below:

1. Right-click the LiveGroup node, or click the wrench  icon in the **Parameters** tab, and select **Edit Contents**.
The LiveGroup icon changes from gray to yellow to indicate the editable state.



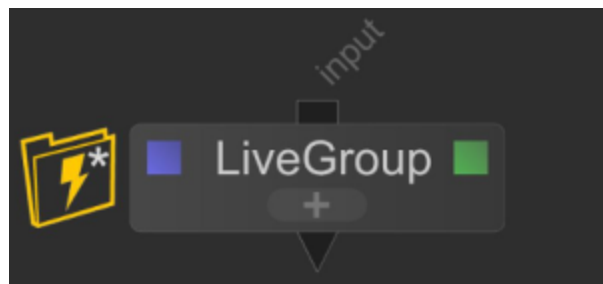
2. Once the LiveGroup is editable, the parameters of contained nodes can be edited and nodes, or node connections, can be added or removed.

Reverting an editable LiveGroup node back to a locked state is necessary if you want to lock down the source and prevent changes from being made to the file or asset. To revert an unlocked LiveGroup node back to a locked state, follow the instructions below:

1. Right-click the LiveGroup node, or click the wrench  icon in the **Parameters** tab, and select **Revert** from the menu.
The contents of the LiveGroup node are locked and can no longer be edited. The LiveGroup icon changes from yellow back to gray again.
2. If the node has unsaved changes, you must confirm whether you want to proceed. If the changes are not published, and you continue to revert the node, the contents are loaded from the selected LiveGroup source.

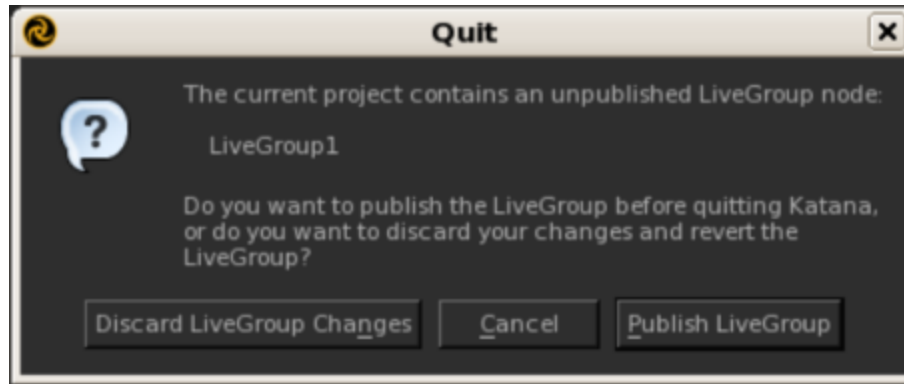
Modified State of Editable LiveGroup Nodes

When changes are made to the parameters or contents of an editable LiveGroup node, the LiveGroup node's icon is drawn with an asterisk, indicating that it has been modified.



The asterisk icon is also shown in the title bar of the Group bubble, which is shown when clicking the LiveGroup node's + button.

When reverting a LiveGroup node or when publishing a LiveGroup node, the modification state of the LiveGroup node is reset and the asterisk disappears. When performing an operation where modifications to a LiveGroup node's parameters or contents would be lost, for example, when creating a new Katana project or when quitting Katana, a dialog message appears asking whether to publish modified LiveGroup nodes to files or assets.



This dialog is also shown when saving the current Katana project, as LiveGroup nodes are not saved in an editable state as part of a Katana project. If you decide not to publish files or assets for modified LiveGroup nodes, those LiveGroup nodes are reverted back to the state they were in when they were made editable.

Publishing a LiveGroup

When a LiveGroup node is editable, a menu option is available to publish changes made to the node's parameters and contents to a LiveGroup source file or asset.

Both editable and non-editable LiveGroup nodes have a source parameter that specifies a LiveGroup source file or asset that the LiveGroup node represents. Publishing a LiveGroup node lets you create a new LiveGroup source file or asset, or lets you create a new version of an existing LiveGroup source asset.

Choose between leaving the LiveGroup node in its editable state after publishing, so you can continue to work on the node, or loading the published LiveGroup source file or asset using its new filename or asset ID.

Publish a LiveGroup

To publish a LiveGroup node, follow the instructions below:

1. Ensure the node is in an editable state. For more information how to make a LiveGroup node editable, refer to [Making a LiveGroup Node Editable](#).
2. After you have made your changes and are ready to publish, right-click on the LiveGroup node, or click the wrench  icon in the **Parameters** tab, and select **Publish...** from the menu.

The **Publish LiveGroup** dialog appears.

3. Enter the filename or asset ID under which you want to publish the LiveGroup source file or asset.
4. Click **Accept**.

The **source** in the **Parameters** tab is updated with the new source filename or asset ID and the node remains in an editable state.



NOTE: If publishing fails for any reason, an error message provides information explaining why publishing changes at that time was not possible.

Publish and Finish Editing Contents of a LiveGroup

To publish and finish editing the contents of a LiveGroup node, follow the instructions below:

1. Ensure the node is in an editable state.
2. After you have made your changes and are ready to publish, right-click on the LiveGroup node, or click the wrench  icon in the **Parameters** tab, and select **Publish and Finish Editing Contents...** from the menu. The **Publish and Finish Editing Contents of LiveGroup** dialog appears.
3. Enter the filename or asset ID under which you want to publish the LiveGroup source file or asset.
4. Click **Accept**.

The **source** in the **Parameters** tab is updated with the new source filename or asset ID, the contents of the LiveGroup node are loaded from the selected source, and the node's contents become locked against editing.



NOTE: If publishing fails for any reason, an error message provides information explaining why publishing changes at that time was not possible, and the node remains in an unlocked state.

LiveGroup Conversion

A LiveGroup node can be converted to a Group node, which brings the contents of the LiveGroup into the current recipe and stops it from updating from the selected source. A Group node can also be converted to a LiveGroup node at any time. A Group that is converted to a LiveGroup is replaced by an editable LiveGroup node with the same contents that the Group node contained.

A LiveGroup node doesn't need to be editable to convert it. To convert a LiveGroup node to a Group node, right-click the LiveGroup node, or click the wrench  icon in the **Parameters** tab, and select **Convert to Group** from the menu. The LiveGroup node is replaced by a Group node with the same parameters and contents, with the exception of the **source** parameters, which is not retained.

To convert a Group node to a LiveGroup node, right-click the Group node, or click the wrench  icon in the **Parameters** tab, and select **Convert to LiveGroup** from the menu. The Group node is replaced by an editable LiveGroup node with the same parameters and contents as the Group node, but with the additional source parameter that allows a LiveGroup node's parameters and contents to be loaded from an external file or asset.

User Parameters and Widget Types

Nodes usually come with a predefined set of parameters that control their behavior. You can also add custom parameters, called User Parameters in Katana. From a node's **wrench** menu, in the **Parameters** tab, choose **Edit User Parameters**. This adds a new group parameter to the node, named **user**, to which you can add custom parameters in the **Parameters** tab. You can add User Parameters, edit their names, rearrange them, and set up additional options and hints for each of them.



NOTE: You can add User Parameters, and UI elements to any node, but they are most commonly used in groups and macros.



NOTE: To automatically create a User Parameter that mimics the format of another parameter, select the node you want to add the parameter to, click on the node's **wrench** icon and choose **Edit User Parameters**. Then middle mouse drag the parameter you want to mimic onto the **Add** menu.

Parameter Types

The types of parameters that can be added as User Parameters on a node are listed in the **Add** menu of a **user** group parameter. The **Add** menu is only shown when editing User Parameters on a node that is edited in a **Parameters** tab. The available parameter types are:

Parameter Type	Description
Number	A single float.
String	A single string.
Number Array	An array of Katana Number variables.
String Array	An array of Katana String variables.

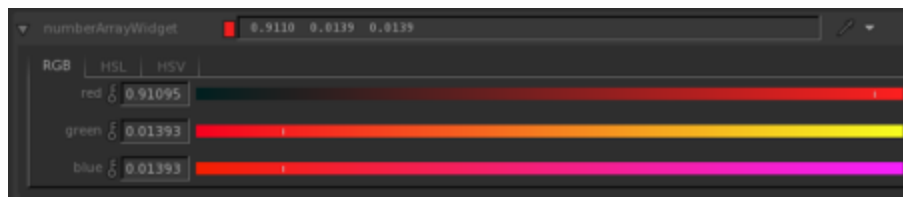
Parameter Type	Description
Float Vector	A text field into which arbitrary text can be entered. The entered text is split at whitespace into text parts, which are then converted to floating point numbers. If the conversion of any part fails, a <code>TypeError</code> exception is raised: <pre>Could not convert string to floatVector</pre>
Color, RGB	A three element array of floats corresponding to the red, green, and blue channel values of an RGB color.
Color, RGBA	A four element array of floats corresponding to the red, green, blue, and alpha channel values of an RGBA color.
TeleParameter	A parameter that references another parameter. The UI provides a drop zone, to drag parameters onto. This creates a Python expression referencing the dropped parameter: <pre>getParam('<name of dropped node>.makeInteractive') .param.getFullName()</pre> In practice this means that a TeleParameter takes on a variable type matching the dropped parameter.
Node Drop Proxy	Node drop proxies appear in the title bar of the parameters for any given node, represented by the  icon. Their purpose is to allow for node drag and drop operations equivalent to the Node Graph tab but within the Parameter tab. The Node Drop Proxy widget within a group's user parameter editor exists to allow a macro to expose drag-from and drop-to operations onto one of its child nodes. It's conceptually similar to a TeleParameter widget, but for node-level drag and drop. A NodeDropProxy allows you to Shift + middle-mouse button drag a node, from the Node Graph onto the NodeDropProxy. It creates an expression to the dropped in node, and gives the user a visual shortcut to that node. This visual shortcut can be dragged onto other nodes, or other group or macro widgets, in the same way that nodes can be dragged from the Node Graph into the Parameters tab (with the exception that the  icon can be left mouse button dragged, rather than Shift + middle-mouse).

Most of the user parameters you can add are data types, but **Group**, **Button**, and **Toolbar** are interface elements you can add to a node:

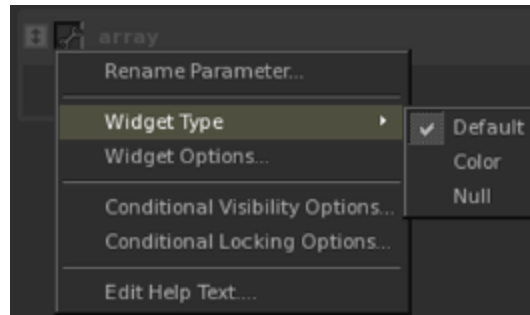
Parameter Type	Description
Group	You can place user adjustable variables, and other interface elements, in a Group inside a Macro's User Interface. You can then apply separate conditional controls to the Group, affecting all elements in it. See <i>Conditional Behavior</i> in the <i>Katana Technical Guide</i> for more on this.
Button	A UI element that runs a script when selected. A button has two Widget Options: A string of text to display on the button. A Python script to run on button press.
Toolbar	A UI element that can contain multiple buttons. Each button has the same display text, and Python script options as a Button user parameter. There are also options to add additional buttons, and control spacing.

Widget Types

For each User Parameter, you can set a widget type that controls how the Katana UI presents that parameter. For example, the image below shows a User Parameter of type **Number Array**, with the widget type set to **Color**. Katana displays the array **user** with the appropriate color component selection widgets.



A number of widget types are available. It's important to note that widget types are closely tied to parameter types. Specific widget types are available for specific parameter types. For example, the **Color** widget type is only available on parameters of type **Number Array** with three or four entries, for RGB or RGBA components respectively. The widget type of a user parameter can be set in the UI, by choosing an item from the **Widget Type** menu in a user parameter's wrench menu in the **Parameters** tab.



NOTE: The widget type of a parameter can also be defined by setting a **widget** hint string. See *Parameter Hints* in the *Katana Technical Guide* for more on this.

Katana also has a dynamic array widget type, added to support shaders with dynamic array parameters. The dynamic array widget type is not accessible through the UI. For more on using dynamic arrays in your shaders see your renderers' documentation.

The Widget Types available in the UI are:

Type	Description	Options
Default	The simplest widget type, consisting of a variable type and data entry field (or fields). The options presented for the Default widget type vary depending on the parameter type the widget type is for.	• Display as Integer - can be True or False. If True, values entered in the widget can only be whole values. This is only available for the Number parameter type. • Drag Minimum - set what the minimum value for the field is allowed to be. This is only available for the Number parameter type. • Drag Maximum - set what the maximum value for the field is allowed to be. This is only available for the Number parameter type. • Drag Sensitivity - set the sensitivity level for the dragging functionality. This is only available for the Number parameter type. • Display Slider - can be True or False. If True, a slider is displayed alongside the entry box. This is only available for the Number parameter type. • Slider Minimum - set the minimum value displayed on the slider. This is only available for the Number parameter type. • Slider Center - set the center value displayed on the slider. This is only available for the Number parameter type. • Slider Maximum - set the maximum value displayed on the slider. This is only available for the Number parameter type. • Slider Exponent - set the exponent value for the slider. This is only available for the Number parameter type. • Expanded By Default - can be True or False. If True, the default functionality is for the parameter to be expanded. This is only available for Group parameter types.

Type	Description	Options
Default	(continued)	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function. • Array Size - set the number of rows and columns that makes up the size of the array. This is only available for array parameter types. • Enable Filmlook Visualization - can be True or False. If True, this option affects what color is shown in the swatch to the right of the parameter name by passing it through the filmlook LUT. This is only available for the Color, RGB and Color, RGBA parameter types. • Restrict Components: 0-1 - can be True or False. If True, it clamps the RGB components to values between 0 and 1. • text - specify the button label. This is only available for the Toolbar parameter type. • icon - specify the file for the use of an icon. This is only available for the Toolbar parameter type. • scriptText - specify the script that is run when the user clicks the button in the widget. This is only available for the Toolbar parameter type. • flat - choose whether the button is flat or has a raised, beveled edge. This is only available for the Toolbar parameter type. • External Editor - opens an external editor for writing the script in a separate window. When you're finished writing the script, save and close the editor; it is automatically added to the Script field. This is only available for the Toolbar parameter type.

Type	Description	Options
Scenegraph Location	Offers the user a dialog box to browse for a scene graph location, and stores the selection. Can be used to pass a selected scene graph location to another widget, or to link directly to a node in the group / macro.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
Scenegraph Locations	Offers the user dialog boxes to browse for multiple scene graph locations. Can be used to pass the selected scene graph locations to another widget, or to link directly to a node - or nodes - in the group / macro.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
CEL Statement	Offers the user a dialog box to add a CEL statement, and stores the statement entered. Can be used to pass the entered CEL statement to another widget, or to link directly to a node in the group / macro.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
Resolution	Offers the user a dropdown menu of the pre-defined render resolutions, and stores the selection. Can be used to pass the selected resolution to another widget, or to link directly to a node in the group / macro.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.

Type	Description	Options
Asset	Offers the user a dialog box to browse for an asset by asset ID, and stores the selection. Can be used to pass the selected asset ID to another widget, or to link directly to a node in the group / macro.	• Sequence Listing - can be True or False. If True, the Sequence Listing option in the file browser is checked, displaying sequences of files as a single item. • Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function. • Tabs To Show - a checkbox option, the results of which are dependent on your asset control system. • Asset type tags - a text entry field to enter lists of tags associated with the volume. For example, geometry volume. • File Type Filter - a text entry field to enter lists of file types to filter. For example, exr pix. • Context - specify a hint that's passed to the Asset plug-in's delegate for browsing for an asset. This allows you to show different assets, depending on whether you are saving a look file or a live group, for example. • Image Viewer - a text entry field to enter the desired image viewer for asset IDs entered into this widget. Defaults to the image viewer specified for the host system, for example, eog. • Force file - can be True or False. If True, the file type is forced to be that specified.

Type	Description	Options
File Path	Offers the user a dialog box to browse for an external file, and stores the selection. Can be used to pass the selected file path to another widget, or to link directly to a node in the group / macro.	• Sequence Listing - can be True or False. If True, the Sequence Listing option in the file browser is checked, displaying sequences of files as a single item. • Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
Boolean	Offers the user a dropdown menu, with Yes and No options, and stores the selection. Can be used to pass the selected Yes or No answer to another widget, or to link directly to a node in the group / macro.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
Popup Menu	Used to define a dropdown menu of either all strings, or all numbers. Returns the string or number shown in the selected menu entry.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function. • Choices - used to add or delete entries from the pop-up menu. Add entries by selecting Add > New Entry.
Mapping Popup Menu	Used to define a dropdown menu, and return a different value to the one shown in the selected menu entry. For example, to show a Yes / No Boolean menu, but return 0 or 1, map the numbers 0 and 1 to the string menu entries Yes and No.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function. • Popup Options - used to add or delete entries from the pop-up menu, and set mapping. Add entries by selecting Add > New Entry. Each entry has two fields, the field displayed in the pop-up menu, and the field returned when the menu option is selected. The field displayed in the menu is the left-most one.

Type	Description	Options
Check Box	Returns a Boolean value, depending on whether the box is checked or not. Used as a value in conditional arguments on other widgets (such as Conditional Visibility, and Conditional Locking).	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
TeleParameter	A parameter that references another parameter. The UI provides a drop zone, to drag parameters onto. This creates a Python expression referencing the dropped parameter. In practice this means that a TeleParameter takes on a variable type matching the dropped parameter.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
Color	Offers the user a color picker dialog, with tabs for RGBA, HSL, and HSV color schemes.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function. • Enable Filmlook Visualization - can be True or False. If True, this option affects what color is shown in the swatch to the right of the parameter name by passing it through the filmlook LUT. This is only available for the Color, RGB and Color, RGBA parameter types. • Restrict Components: 0-1 - can be True or False. If True, it clamps the RGB components to values between 0 and 1.
Multi	Mimics the structure of a CameraCreate node's screenWindow parameter. Unlike a screenWindow, a Multi is created empty, and you have to add - and name - the individual values manually.	• Expanded By Default - can be True or False. If True, the default functionality is for the parameter to be expanded. This is only available for Group parameter types. • Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.

Type	Description	Options
Null	Hides the parameter from the Parameters tab.	• Expanded By Default - can be True or False. If True, the default functionality is for the parameter to be expanded. This is only available for Group parameter types. • Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.
Script	Runs a preset script when the user clicks the button in the widget.	• Button Text - specify the button label. • Script - specify the script that is run when the user clicks the button in the widget. • External Editor... - opens an external editor for writing the script in a separate window. When you're finished writing the script, save and close the editor; it is automatically added to the Script field.
Script Editor	Offers the user a script entry box for creating their own scripts to run.	• Constant (No Animation) - can be True or False. If True, values entered in the widget are keyable. • Display Only - can be True or False. If True, the widget displays, but without function.

Widget Type Availability

Widget type availability is dependent on the User Parameter type. The available widget types for each variable type are:

User Parameter	Available Widget Types
Number	Default, Boolean, Popup Menu, Mapping Popup Menu, Check Box, Null.
String	Default, Scene Graph Location, CEL Statement, Resolution, Asset, File Path, Boolean, Popup Menu, Mapping Popup Menu, Script Button, Check Box, TeleParameter, Script Editor, Null.

User Parameter	Available Widget Types
Group	Default, Multi, Gradient, Null.
Number Array	Default, Color, Null. Note: The Color widget type is only available on arrays containing 3 or 4 values (for RGB, and RGBA respectively).
String Array	Default, Scene Graph Locations, Null.
Float Vector	Default, Null.
Color, RGB	Default, Null.
Color, RGBA	Default, Null.
Button	Default, Scene Graph Location, CEL Statement, Resolution, Asset, File Path, Boolean, Popup Menu, Mapping Popup Menu, Script Button, Check Box, TeleParameter, Script Editor, Null.
Toolbar	Default, Scene Graph Location, CEL Statement, Resolution, Asset, File Path, Boolean, Popup Menu, Mapping Popup Menu, Script Button, Check Box, TeleParameter, Script Editor, Null.
TeleParameter	Default, Scene Graph Location, CEL Statement, Resolution, Asset, File Path, Boolean, Popup Menu, Mapping Popup Menu, Script Button, Check Box, TeleParameter, Script Editor, Null.
Node Drop Proxy	Default, Scene Graph Location, CEL Statement, Resolution, Asset, File Path, Boolean, Popup Menu, Mapping Popup Menu, Script Button, Check Box, TeleParameter, Script Editor, Null.

Collections and CEL

Overview

Collection Expression Language, or CEL, is used to describe the scene graph locations on which an operation or assignment acts. CEL statements can also be used to define **collections** which may then be referenced in other CEL statements.

There are two different purposes that CEL statements are used for: matching and collecting.

Matching is the most common operation, and is used as scene graph data is generated. Many nodes in Katana have CEL statements that allow the user to specify which locations the operation defined by this node act on. For instance, CEL statements are used in MaterialAssign nodes to specify which locations in the hierarchy have a particular material assigned to them. As each scene graph location is generated it is tested against the CEL statement to see if there is a match. If it is, the operation is executed at that location. This matching process is generally a very fast one to compute.

Collection is a completely different type of operation, a CEL statement used to generate a collection of all locations in the scene graph that it matches. Depending on the CEL statement this can potentially be expensive as to evaluate it may have to open every location in the scene graph to check for a match. Collecting is usually done as part of a baking process or to select things in the UI ('Find and Select'), but also has to be done for light linking if you use an arbitrary CEL expression to specify the lights.

CEL in the User Interface

In the UI, a standard **CEL Widget** interface is provided for CEL expressions. For your convenience, this allows you to build CEL expressions out of three different types of components (called statements):

- Paths – these are explicit lists of scene graph paths. If you drag and drop locations from the **Scene Graph** tab onto the 'Add Statements' area of the CEL widget you are automatically given a CEL expression that based on those paths.
- Collections – a pre-defined named collection of scene graph locations. Essentially these are arbitrary sets of locations that are handed off for use downstream in the pipeline. Collections can be created in a Katana scene using the CollectionsCreate node, and can also be passed from one Katana project to another using Look Files.

- Custom – these allow complex rule based expressions, such as using patterns with wildcards in the paths, or 'value expressions' that specify values that attributes must have for matches.

CEL expressions can also be built out of combinations of these components using union, difference, and intersection operations.

Guidelines for using CEL

Using CEL to Specify Light Lists in the LightLink Node

There is only one node that does a collect operation while actually evaluating the Katana recipe: the LightLink node.

LightLink allows you to use a CEL statement to determine which lights to link to, which allows a lot of flexibility in selecting which lights are linked, but involves running a collection operation at runtime. How the CEL statements are used to specify the lights (and where those lights are in the scene graph) should be set up carefully to maximize efficiency and avoid having to evaluate too many scene graph locations. In general it is most efficient to use a list of explicit paths for the light list. If you need to use more general CEL expressions, such as those that use wild cards, it is best to make sure these only need to run to a limited depth in the scene graph. The worst case is an expression with recursion that potentially needs every scene graph location to be tested.

'Find and Select' Isn't a Good Test for Efficiency

It's wrong to run a 'Find and Select' to test the efficiency of a CEL statement that is only going to be used for matching. For instance, the CEL statement `//myGeoShape` that only matches with locations called 'myGeoShape' is very fast to run as a match when evaluating any location, but takes a very long time to collect because it has to expand the whole scene graph looking for locations with that name.

Make CEL Statements as Specific as Possible

The expense is generally in running operations at nodes rather than evaluating if a location matches a CEL expression, so it's good make sure that nodes only run on the locations really necessary.

For instance: it's most efficient if a CEL statement can be made to only run on the correct locations, based on looking at the path name of the location. If the attribute values have to be tested, such as tests for the type of location, these tests are more expensive, as it requires the attributes at that location to be calculated as well.

Another typical case is using the CEL expression `//*`, which is a very fast expression to match but usually means that a node runs at far more locations than it needs to.

Avoid Extensive Use of Deep Collections

Collections brought in by a Look File are defined at the root location that the Look File is assigned to. If those collections are only used in the hierarchy under the location they are defined at evaluation is efficient. However, if you refer to that collection in other parts of the scene graph then there is a cost as the scene graph has to be evaluated at the location the collection is defined.

An example where this can be a problem is if you've got collections defined at **/root** that reference a lot of other collections defined deeper in the scene graph. This means that to just evaluate /root you need to examine the scene graph to all those other locations as well.

Avoid Complex Rules in Collections at /root

Collections of other collections are useful and are efficient if all the collections are defined using explicit paths. If these collections are created using more complex rules, in particular recursive rules, you can run into efficiency problems.

Avoid Using '*' as the Final Token in a CEL Statement

There are optimizations in CEL to first compare the name of the current location against the last token in the CEL statement. For this reason, it's advisable to have a specific last token in a CEL statement, instead of using '*' as a wildcard. For instance, if you've got a rule to only run on geometry locations that end with the string 'Shape' it's more efficient to have a cell expression such as:

/root/world/geo/*Shape

rather than

/root/world/geo/*.

Paths vs Rules

CEL has a number of optimizations for dealing with explicit lists of paths. This means using paths are the best way of working in many cases, and matching against paths is generally very efficient as long as the list of paths isn't too long.

As a general rule it's more efficient to use explicit lists of paths than active rules for up to around 100 paths. If you have explicit lists with many thousands of paths you can run into efficiency issues where it may be very worthwhile using rules with wild-cards instead.

'Select and Collect' operations are always more efficient with an explicit path list.

Use Differences Between CEL Statements Cautiously

Taking the difference between two CEL statements can be expensive. In particular if two CEL statements are made up of paths when you take the difference it's no longer treated as a simple path list so won't use the special optimizations for pure path lists.

Single nodes with complex difference based CEL statements can often be more efficiently replaced by a number of nodes with simpler CEL statements.

Working with Attributes

Overview

At its core, everything in Katana is about creating and manipulating attributes. These attributes, stored at locations within the scene graph, represent the information a renderer needs to render a scene, such as geometry data, transforms, or what material should be applied at a given location.

Although almost all nodes, in essence, manipulate attributes, Katana provides a number of special, general-purpose nodes that give you free reign to create or manipulate the values of any attribute at one or more locations. The most common are `AttributeSet`, `AttributeScript`, and `OpScript`.

- The `AttributeSet` node is used to create, override, or delete attributes at one or more locations.
- The `AttributeScript` node is used to run Python scripts at specified locations. The script can access the attributes of the location (and others) and use Python to make changes.
- The `OpScript` node is a Lua-based interface to the Op API, which is both faster and more powerful than `AttributeScript`. In particular, the `OpScript` node also allows you to modify the structure of the scene graph hierarchy, such as deleting locations or creating new child locations. For more information about `OpScript` and the Op API, please refer to the *Op API* chapter in the *Katana Technical Guide*.



NOTE: Though the `AttributeScript` node is still available for use within Katana, the `OpScript` node is recommended in favor of it. Using the `OpScript` node instead of the `AttributeScript` node provides many improvements and additional features that were not possible with `AttributeScript`. However, the `AttributeScript` node still exists and you can use it, if you choose, particularly if you want to run old Katana projects.

To learn how to manipulate attributes, refer to either:

- [OpScript Nodes](#)
- [AttributeSet and AttributeScript Nodes](#)
- [OpScript vs AttributeScript](#)

OpScript Nodes

The OpScript node allows you to use the Lua scripting language to manipulate attributes at a single location, or at multiple locations. Technically, the OpScript node provides Lua bindings for the C++ Op API, so what you can do with the Op API, you should also be able to do with the OpScript node. Lua is also multi-threaded, which makes it very fast.

The OpScript node has many exciting features that make it very powerful:

- Overwrite, create, and delete attributes at any scene graph location.
- Accept multiple inputs.
- Create and delete child scene graph locations.
- Copy scene graph locations.
- Use Lua bindings for Op API C++ functions.

The OpScript node uses CEL (Cel Expression Language) to specify the locations where the script runs. The OpScript node can be used to read attributes on any scene graph location, edit attributes at the specified scene graph locations, create child locations, delete child locations, and copy scene graph locations. When running on multiple locations, a script runs separately at each location, so targeting 100 locations means that your OpScript runs 100 times.

Since there are Lua bindings for the Op API, you may only need the Op API if you really need the speed and efficiency of C++, meaning many powerful tools can be written with the OpScript node, and potentially wrapped up inside a macro for other users. If you do need to use the Op API, then the Lua interface allows for easier prototyping before fully committing to C++, which is especially useful for proof of concept.

Adding an OpScript

To add an OpScript node to a recipe:

1. Create an OpScript node and connect it to the recipe at the point you want to insert the Lua script.
2. Select the OpScript node and press **Alt+E**.
The OpScript node becomes editable within the **Parameters** tab.
3. Assign the scene graph locations (or target nodes) this Lua script is to run on to the **CEL** parameter (see [Assigning Locations to a CEL Parameter](#)).
4. Select when to run the script using the **executionMode** dropdown:
 - **immediate** - runs the script immediately.
 - **deferred** - runs the script at a later node in the node graph, as specified by the **applyWhen** parameter.
5. If you chose **immediate**, the **applyWhere** dropdown is available for you to choose where the script is run:
 - **at all locations** - at all the locations in the node graph.

- **at specific locations** - at only the location specified by the **location** parameter. If this location doesn't exist, it is created automatically.
- **at locations matching CEL** - at only those locations in the node graph that match the CEL statements.

If you chose **deferred**, the **applyWhen** dropdown is available for you to choose when the script is run, if not immediately:

- **during op resolve** - the script and its arguments are added as attributes to be executed later by an OpResolve node.
- **during material resolve** - the script and its arguments are added as attributes under the **material.scenegraphLocationModifiers** group attribute.
- **during katana look file resolve** - the script and its arguments are added as attributes under the Scene GraphLocationModifiers group attribute and are evaluated by a LookFileResolve node or by the first implicit resolver if no LookFileResolve node is present.



NOTE: Each of the **applyWhen** options has additional parameters that are available, depending on which you choose. Refer to the *Katana Reference Guide* for more information on each of these options and what they do.

6. If you chose **immediate**, set the **inputBehavior** dropdown. This controls how input ports on the node are mapped onto the inputs of the underlying Op, and is only meaningful when the node has one or more **invalid input ports** - a port that is not connected to an output port or is connected to an output port that doesn't provide data.

If you chose **deferred**, set the **modifierNameMode** dropdown to either:

- **node name** - deferred OpScripts are added as group attributes within the "Ops" group, and the name of the node is used for the sub-group. Since node names must be unique in project, the resulting attribute name can change.
 - **specified** - use a fixed name for the OpScript sub-group.
7. In certain instances, such as when **executionMode** is set to **immediate**, or when it is set to **deferred**, **during op resolve**, the **resolveIds** parameter displays. Specify the **resolveIds** in the form of "resolveName1", "resolveName2" and so on. This ensures the OpScript is resolved only by the op resolvers that contain a matching resolve ID.
 8. If you choose **deferred**, and set the **applyWhen** parameter to either **during op resolve** or **during katana look file resolve**, you can set the **recursiveEnable** parameter. Enabling this parameter results in the script being run at every location beneath the assigned locations. This is far more efficient than using the equivalent recursive **CEL** statement.
 9. If you want to set the OpScript node to have multiple inputs, tick the **Display as multi-input** box. By default, this box isn't ticked.
 10. Finally, enter the Lua script in the **script** parameter.



TIP: To have an OpScript, running at multiple locations, use the same stable, random numbers at each location, the **math.randomseed()** and **math.random()** functions can be used. OpScript's



math.randomseed() and **math.random()** implementations use re-entrant pseudo-random number generation functions, and implement the default flavors available in Lua.

```
local seed = math.randomseed()
local randVal_01 = math.random(1, 10)
local randVal_02 = math.random(11, 20)
local randVal_03 = math.random(21, 30)
```

OpScript Tutorials

Please note that the following examples are located in the **Help > Example Projects** menu in Katana, or within the katana install directory at \$KATANA_HOME/demos/katana_files/opscript_tutorial.katana.



TIP: Refer to [Appendix D: AttributeScript to OpScript Conversion](#) on page 410 for more information about switching to OpScript, with examples showing the Python to Lua conversions.

Creating Scene Graph Locations

Using CreateChild()

This example shows you how to create child locations using the OpScript node. By default, if you specify **Interface.CreateChild('childName')** inside your OpScript without specifying the **opType** parameter, your child location inherits the opType used by the parent. So, in this case, it recursively uses this opType, OpScriptLua, and creates an infinite number of child locations.



NOTE: Please refer to the *OpScript Reference Guide* for more information on any of the following functions, or for information on exposed functions for the OpScript.

There are a few solutions to this: check if we match the CEL, create a hierarchy based on an if-else statement, or use a StaticSceneCreate op to create a hierarchy.

Check If We Match the CEL

A simple way to get around this is to use the following command:

```
if Interface.AtRoot() then
    Interface.CreateChild("child_a")
end
```

In the OpScript node, set the CEL statement to **/root/world** by previously creating a **/root/world** location using **LocationCreate**. Also make sure that the **applyWhere** parameter is set to **at locations matching CEL**.

Create a Hierarchy Based on if-else Statement

We create a hierarchy by checking which location we are currently at and executing the commands associated to that if statement. When this script is run for the first time, it creates a child location at **/root**, which is **/root/world**. Then, when the OpScript is executed at the child location, it creates **/root/world/geo**. This continues until the last condition is met. In this way, we avoid infinite recursion. We've set the **applyWhere** parameter to **at all locations** too, so that we don't need to worry about specifying **/root**.

```
path = Interface.getOutputLocationPath()

if path == "/root" then
    Interface.CreateChild("world")
elseif path == "/root/world" then
    Interface.CreateChild("parent")
elseif path == "/root/world/parent" then
    Interface.CreateChild("child_a")
elseif path == "/root/world/parent/child_a" then
    Interface.SetAttr("test", IntAttribute(123))
end
```

Use a StaticSceneCreate Op to Create a Hierarchy

We can use the **StaticSceneCreate** op to create a hierarchy without the use of if-else statements using the **OpArgsBuilder.StaticSceneCreate()** function. You can input the following directly into an OpScript node, or refer to the example OpScript Tutorial file. Again, we've set the **applyWhere** parameter to be run **at all locations**.

```
sscb = OpArgsBuilders.StaticSceneCreate(true)
sscb:createEmptyLocation("/root/world/parent/child_a", "group") --create a scenegraph
location with a type
sscb:setAttrAtLocation("/root/world/parent/child_a", "test_id", IntAttribute(123)) --
set the attribute
Interface.ExecOp("StaticSceneCreate", sscb:build()) --execute the Op
```

Deleting Scene Graph Locations

There are three methods available to remove scene graph locations: **deleteChild()**, **deleteChildren()** and **deleteSelf()**. By allowing you to remove your own scene graph locations, you can re-implement your own Prune node, for instance. This tutorial looks at each of these methods, but please refer to the OpScript Tutorial Example Projects for context. You can comment or uncomment the relevant commands.

Delete the Child by Name

The OpScript node allows you to delete newly-created children and incoming child locations, like so:

```
Interface.DeleteChild("child_a")
```

Note that `child_a` is the immediate child of the matching CEL. For example, if your CEL statement looked like this:

```
/root/world/geo/parent
```

and the children that exist here are

```
/root/world/geo/parent/child_a and /root/world/geo/parent/child_a/grandchild_a
```

the **DeleteChild()** function cannot delete `grandchild_a`.

Delete All Children

Deleting all children under which the OpScript is being cooked at is straightforward. This also deletes all newly-created children and incoming ones:

```
Interface.DeleteChildren()
```

Delete Self

You are also able to delete the current output location using **Interface.DeleteSelf()**, however, all calls to **DeleteSelf()** keep the location in its parent's potential children list. So, it's advisable to use **DeleteChild()** instead, if possible.

Copying Scene Graph Locations and Attributes

Another useful feature of the OpScript node is the ability to copy scene graph locations. For instance, allowing you to re-implement the HierarchyCopy node, if you wish. You can achieve this using **Interface.CopyLocationToChild()** function. Please refer to the OpScript Tutorial Example Projects and use the following code in conjunction for understanding the process.

Copying Scene Graph Hierarchies

These tutorials have shown how you can copy hierarchies very easily using the **CopyLocationToChild()** function. Using the following piece of Lua code, we can copy the `/root/world/geo/parent_a` hierarchy to the locations matching the CEL statement provided, in this case, `/root/world/geo`. The result is another hierarchy at `/geo` with `/root/world/geo/parent_b/child_a`. The resultant hierarchy has all the attributes copied over too.

```
if Interface.AtRoot() then
    Interface.CopyLocationToChild("parent_b", "/root/world/geo/parent_a")
end
```

Copying Attributes Across Different Inputs

Since the attributes are being copied over along with the scene graph locations, we can use this to our advantage. As the OpScript node supports multiple inputs, you can effectively recreate a custom Merge or Switch node, again, increasing the potential of the OpScript node.

Using the same function as above, look at how you can copy from one input to another. Please refer to the OpScript Tutorials example file for context.

We have two LocationCreate nodes and an AttributeSet node corresponding to each LocationCreate node. The two node graph branches are then connected to separate input ports of the OpScript node. The **CopyLocationToChild()** function that we used in the previous example has two extra arguments that we haven't explicitly specified; they are: the input index and the order of where you want to place your new hierarchy.

```
if Interface.AtRoot() then
    Interface.CopyLocationToChild("target_child_a",
    "/root/world/another_parent/another_child_a", 1, "child_a")
    -- Interface.CopyLocationToChild("target_child_a",
    "/root/world/parent/child_a", 0, "child_a")
end
```

This script copies over the name attribute from the second input of the OpScript to the target_child_a location. The name changes depending on which line you comment/uncomment.

AttributeSet and AttributeScript Nodes

OpScript vs AttributeScript

The OpScript node is a new feature of Katana 2.0. In many ways it is similar to the existing AttributeScript node in that it allows you to write scripts using nodes directly in the UI which can inspect and modify attributes, but it is designed to also provide a lot more functionality and to work efficiently with the new Geolib3 architecture. The old AttributeScript node is still there, mainly to allow old Katana scenes to still run, but going forward we strongly recommend using the new OpScript node.

Where the AttributeScript node has Python syntax, the OpScript node uses the Lua scripting language instead. This was chosen due to Lua's capabilities to run inside other processes and on multiple threads, which makes it a good match for the new Geolib3 architecture. It also comes with Lua bindings for the C++ OpAPI, making it extremely powerful since you have direct access to the functions normally used to write C++ Ops. What you can do with the OpAPI, you can also do here, without any behavioural differences.

The OpScript node was introduced because of the multi-threading limitations of Python. In particular, Python's Global Interpreter Lock (GIL) can block multi-threaded processing. To avoid this problem we run AttributeScripts in a

separate process pool, but this means that there is an additional inter-process communication overhead. The OpScript node is a viable and stronger alternative to the AttributeScript node, due to Lua's multi-threading nature.

Since the OpScript node is aimed at replacing the AttributeScript node, you may find that all the common operations that you can do with AttributeScripts, you can also do with OpScripts. The OpScript node runs a Lua script on a single location, or multiple locations in the scene graph to create, delete, copy and manipulate attributes.

OpScripts are very powerful and you can do much more than what you can do with AttributeScripts. In particular you can manipulate the scene graph hierarchy as well as manipulate attributes. The main things to note are:

The ability to manipulate scenegraph locations. In addition to reading and editing attributes at a given scenegraph location, you are able to create child locations, delete child locations and copy scenegraph hierarchies.

OpScript nodes have the ability to accept multiple Node Graph inputs. This allows you to read attributes at various different Node Graph branches and gives you much more flexibility when manipulating attributes.

The above capabilities opens the door to many opportunities that would not have been possible with AttributeScripts. For example, because you're able to delete child locations, you can effectively recreate your own version of the Prune node. Or maybe you want to create a custom Merge operation: having multiple inputs accepted by the OpScript node, makes this possible now. Other examples of nodes that can be recreated with the OpScript node are: HierarchyCopy, Switch and Isolate nodes.

Making Changes with the AttributeSet Node

To add an AttributeSet node to a recipe:

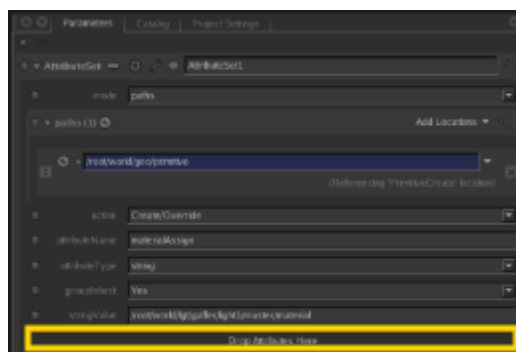
1. Create an AttributeSet node and connect it to the recipe at the point you want to make the change.
2. Select the AttributeSet node and press **Alt+E**.
The AttributeSet node becomes editable within the **Parameters** tab.
3. Select the assignment mode from the **mode** dropdown:
 - **paths** - the locations influenced by this node are selectable by their path.
 - **CEL** - the locations influenced by this node are selectable using CEL.
4. Assign the locations to influence with this node to either the **paths** or **celSelection** parameter (depending on your selection in step 3).
5. Select what type of action this node is performing:
 - **Create/Override** - adds a new attribute or overrides an existing one.
 - **Delete** - if it exists, removes an attribute from the location.
 - **Force Default** - forces the attribute back to its default.
6. Enter the name of the attribute to influence in **attributeName**.
You can enter a grouped attribute by separating the parts of the attribute with a . (full stop), for instance **geometry.point.P**.

If the **action** parameter is **Create/Override**:

7. Select the type of the attribute using the **attributeType** dropdown.
8. With the **groupInherit** parameter, select whether you want the attribute changes to be inherited by any scene graph children. For instance, a new attribute on **/root/world/geo** created with this option set to **Yes** is inherited by all children of **/root/world/geo**.
9. Enter the new attribute value in the **<type>Value** parameter, for instance, **stringValue** for a string.



TIP: It is possible to middle-click and drag from an attribute in the **Attributes** tab to the **Drop Attributes Here** hotspot in an AttributeSet node's **Parameters** tab to automatically create a field to set the dragged attribute.



Adding an AttributeScript

To add an AttributeScript node to a recipe:

1. Create an AttributeScript node and connect it to the recipe at the point you want to insert the Python script.
2. Select the AttributeScript node and press **Alt+E**.
The AttributeScript node becomes editable within the **Parameters** tab.
3. Assign the scene graph locations (or target nodes) this Python script is to run on to the **CEL** parameter (see [Assigning Locations to a CEL Parameter](#)).
4. Select when to run the script using the **applyWhen** dropdown (see [When to Run](#)):
 - **immediate** - run the script immediately.
 - **during attribute modifier resolve** - run the script when other attribute modifier plug-ins are being resolved.
 - **during katana look file resolve** - run the script when any katana look files are resolved (or would be resolved if there are none in the recipe).
 - **during material resolve** - run the script when material resolving occurs.
5. If you want to run an initial script before the main script, select **Yes** in the **initializationScript** parameter (see [Using Initialization Scripts](#)).
If **Yes** is selected, a **setup** parameter is displayed. Enter your initialization script in the **setup** parameter.
6. Finally, enter the Python script in the **script** parameter.

Using Python within an AttributeScript Node

An AttributeScript node is a node in the node graph that runs a Python script on a location, or multiple locations, in the scene graph to manipulate attributes. It can read attributes from any location in the scene graph, and can change attributes on the scene graph locations it runs at. A single AttributeScript can run at multiple scene graph locations for actions such as assigning the same material to multiple geometry objects. When running at multiple locations, a script runs separately at each location. So, if you target 500 or 1000 scene graph locations with an AttributeScript node, your AttributeScript runs 500 or 1000 times.

The key to understanding how an AttributeScript works, and what it can or can't do, is understanding how a Katana project is constructed and traversed. Katana projects are assembled in the node graph, which is made up of interconnected nodes (a recipe). Scenes are rendered from the scene graph, which is made up of cascading locations. The contents of the scene graph are determined by the nodes, parameters, and interconnections in the node graph. You'll have noticed that while you can manually edit parameters on nodes in the node graph, you can't edit attributes on a scene graph location. This is because the flow of information from node graph to scene graph is strictly one-way. This is an important concept to keep in mind when using AttributeScript nodes. An AttributeScript runs in the scene graph, at the locations you set it to, not in the Node Graph. For this reason, an AttributeScript cannot use the Katana Python APIs available elsewhere in Katana (such as the Nodegraph API, Farm API, or UI4).

An AttributeScript can read data from any scene graph location, and set data at the scene graph location it runs at. It cannot read data from a node in the Node Graph, or change parameters on a node in the node graph. When you use an AttributeScript to read an attribute, such as the **type**, from a scene graph location, what you're doing is using Python to access the result of cooking the Node Graph recipe, not accessing the parameter on the Node Graph node.



NOTE: As an AttributeScript cannot use the NodegraphAPI, and as scene graph locations are created from nodes in the Node Graph, an AttributeScript cannot create new scene graph locations, or delete existing ones.

Understanding Locations

By default, an AttributeScript reads attributes from the scene graph locations it runs at. However, it is possible to read attributes from any scene graph location by setting the **atLocation** flag, followed by the location to be queried. For example, to have a script running at location **/root/world/geo/primitive1** read the type at location **/root/world/primitive2** use the following:

```
GetAttr( "type", atLocation="/root/world/geo/primitive2" )
```

Understanding Inheritance

The value returned when querying an attribute is the value at the location the AttributeScript is running, or the specified location if the **atLocation** flag is set, not the value at any parent locations. If an attribute is not set on the target location, an AttributeScript that queries that attribute returns **None**, even if the desired attribute is set on a parent location. It is possible to read values from parent locations by setting the inherit flag to **inherit=True**. For example, to have a script running at location **/root/world/geo/primitiveGroup/primitive1** read an attribute named **user.string** inherited from its parent location use the following:

```
GetAttr( "user.string", inherit=True )
```



NOTE: You can use the inherit flag in conjunction with the **atLocation** flag, to get values on the parent location of the one specified using the **atLocation** flag.

Using Initialization Scripts

If you have an AttributeScript running at multiple locations, and there are operations in your script that return the same result for each location, having them run separately every time is not efficient. Katana offers the option of running an initialization script before an AttributeScript which, unlike the main AttributeScript, runs only once. Use an initialization script by setting an AttributeScript node's **initializationScript** parameter to **Yes**, and entering a script in the resulting script box. For example, you could calculate transformation matrices to be applied to all locations an AttributeScript runs at, and have the calculations performed in an initialization script just once, then used at each location.

AttributeScripts use the **user** module to pass variables from initialization script to the main script. For example, were the code below run as an initialization script, variable **a** would be available in the main script, while variable **b** would not.

```
user.a = 1
b = 2
```



NOTE: Variables stored in the user module are namespaced in the main script, so to access variable **a** in the main script enter:

```
user.a
```

To have an AttributeScript running at multiple locations use the same stable random numbers at each one, use an initialization script to generate the random numbers:

```
seed = ExpressionMath.stablehash( GetFullName() )
user.randVal_01 = ExpressionMath.randval( 1, 10, seed )
user.randVal_02 = ExpressionMath.randval( 11, 20, seed )
user.randVal_03 = ExpressionMath.randval( 21, 30, seed )
```

When to Run

The Katana scene graph has a nested tree structure, and at render time the tree is traversed starting at the **/root** node, and working down. If a location has an `AttributeScript` attached to it, by default, the script is run immediately, as soon as the render process reaches that location. You can set an `AttributeScript` to defer running until a predetermined point in the render process by selecting an option other than **immediate** from the **applyWhen** menu. The options available are:

- immediate
- during attribute modifier resolve
- during katana look file resolve
- during material resolve

For example, an `AttributeScript` reading material attributes needs to run at material resolve, otherwise the attributes won't be there to read. In that case, set the **applyWhen** parameter to **during material resolve**.

Getting Multiple Time Samples

Some render effects, particularly motion blur, rely on taking more than one sample per frame. The number of samples taken is determined by the **maxTimeSamples** parameter, with the samples spread evenly between the given **shutterOpen** and **shutterClose** times. It's possible to query an attribute's value at every sample time. These multiple time samples are returned as data type `ScenegraphAttr`, and take the form of a dictionary where the keys are float values, representing the times sampled. To read a multiple time sampled attribute, use **GetAttr().getSamples()** with the flag **asAttr=True**. For example, to read a location's X, Y, and Z scale values as a multi-time sampled `ScenegraphAttr` enter the following in an `AttributeScript`:

```
scaleMulti = GetAttr( 'xform.interactive.scale',
asAttr=True ).getSamples()
```

If you print the result, assuming the scale is animated over the sample range, the result looks something like this:

```
{-0.25: [1.73, 1.73, 1.73], 0.0: [2.0, 2.0, 2.0], 0.25: [1.90, 1.90, 1.90]}
```



NOTE: As a `ScenegraphAttr` takes the form of a dictionary, the entries are not necessarily ordered. You can generate a sorted list of all the dictionary keys in a `ScenegraphAttr` using **sorted()**. For example, **sortedKeys = sorted(scaleMulti)**

Custom Error Messages with an AttributeScript Node

If a node has an attribute named **errorMessage**, an error is displayed at the node's scene graph location, taking the parameter of **errorMessage** as the message shown. With an `AttributeScript` node, you can set the **errorMessage** attribute on the target node using Python script:

```
SetAttr( "errorMessage", [ "Your error message" ] );
```

Katana steps through the scene graph at render time, and if it encounters a node with the **type** attribute set to "error", the render is stopped at that point. With an AttributeScript node, you can set the **type** attribute with parameter "error" on the target node using Python script:

```
SetAttr( "type", [ "error" ] );
```

Stop Render Error Message

Follow these instructions to create a scene with an AttributeScript node to stop rendering and give an error message:

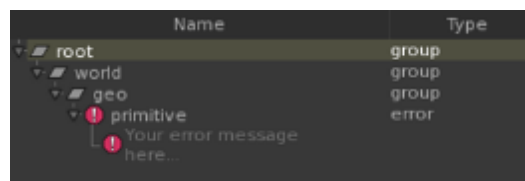
1. Create a primitive using a PrimitiveCreate node.
2. Create a Render node.
3. Connect the output of the PrimitiveCreate node to the input of the Render node.
4. Create an AttributeScript node and place it between the PrimitiveCreate node and the Render node.
5. Select the AttributeScript node and press **Alt+E**.

The AttributeScript node becomes editable within the **Parameters** tab.

6. Assign the attribute script to run at the scene graph location created by the Primitive Create node (see [Assigning Locations to a CEL Parameter](#)).
7. Using the **applyWhen** dropdown set the script to run **immediately**, and set **InitializationScript** to **No**.
8. Enter the following into the **script** parameter:

```
SetAttr( "type", [ "error" ] );
SetAttr( "errorMessage", [ "Your error message here" ] );
```

The PrimitiveCreate node has its **type** attribute set to "error", which halts rendering. In this example, the **errorMessage** attribute is set to "Your error message here" so this message is displayed at node's scene graph location.



Camera Threshold Error Message

Using the available Katana Python APIs, you can recover values from any non-default attribute, at any scene graph location in your recipe. Follow these instructions for using **GetAttr()** in an AttributeScript to get the translation values of the camera, compare those values to the target node, and write an errorMessage if the target node is in front of and within a threshold distance of the camera:

1. Create a primitive using a PrimitiveCreate node.
2. Create a camera using a CameraCreate node.
3. Create a Merge node.

4. Connect the outputs of the PrimitiveCreate, and CameraCreate nodes to the inputs of the Merge node.
5. Create an AttributeScript node.
6. Connect the output of the Merge node to the input of the AttributeScript node.
7. Select the AttributeScript node and press **Alt+E**.

The AttributeScript node becomes editable within the **Parameters** tab.

8. Assign the PrimitiveCreate node as the scene graph location this AttributeScript node is to run on (see [Assigning Locations to a CEL Parameter](#)).

9. Enter the following into the AttributeScript **script** parameter:

```
from numpy import *
import math

# Set the distance threshold
theThreshold = 1.1;

# Get parameters from the camera
camTrans = GetAttr( 'xform.interactive.translate', atLocation =
'/root/world/cam/camera' );

# Get parameters from the target object
objTrans = GetAttr( 'xform.interactive.translate' );

# Declare a numpy array to hold vector from object to camera
objVec = array( [ 0, 0, 0 ] );

# Get vector from object to camera into objVec
objVec[ 0 ] = objTrans[ 0 ] - camTrans[ 0 ];
objVec[ 1 ] = objTrans[ 1 ] - camTrans[ 1 ];
objVec[ 2 ] = objTrans[ 2 ] - camTrans[ 2 ];

# Get the magnitude of the vector from object to camera
objCamDist = math.sqrt( math.pow( objVec[ 0 ], 2 ) + math.pow( objVec[ 1 ], 2 ) +
math.pow( objVec[ 2 ], 2 ) );

# Set the errors if the object is in front of the camera
# and too close.
if objCamDist < theThreshold and objVec[ 2 ] <= 0:
SetAttr( "type", [ "error" ] );
    SetAttr( "errorMessage", [ "Object is too close to the camera" ] );
```

The Python script shown above uses **GetAttr()** to read the translation values of the target object, and - by setting the `atLocation` flag - also reads the translation values of the camera. When the magnitude of the vector from object

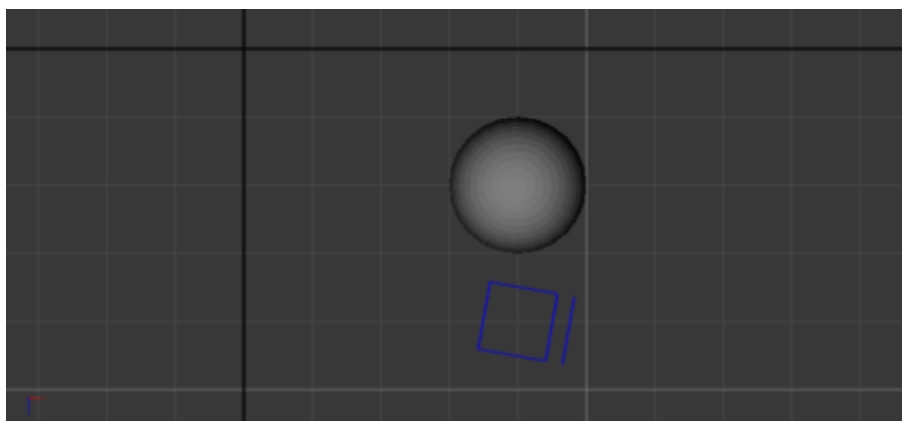
to camera is below the specified threshold, and the object is in front of the camera, the attribute **type** on the target object is set to "error", and the attribute **errorMessage** set to "Object is too close to the camera". Whether the object is in front of the camera or not is found by checking if the Z value of the vector from object to camera is a positive or negative number. This assumes that the camera X, Y, and Z axis are aligned with the world X, Y, and Z axis.



NOTE: Text copied and pasted from a .pdf does not retain the formatting shown. Python code excerpts do not retain indent information.

Camera Alignment Error Message

If the camera is not aligned with the cardinal axis, then add an extra step to compensate. For example, in the scene shown below, the camera is at position (4, 2, 4) with rotation of -100° about its Y axis, and the primitive sphere is at (4, 2, 2).



In the world frame, the sphere is in front of the camera but, in the camera frame, the sphere is behind the camera. In world frame the vector from sphere to camera is (0, 0, -2). To convert this world frame vector to the camera frame, multiply the vector from sphere to camera by the rotation matrix of the camera, using the numpy scientific library for Python.

1. Create a primitive using a PrimitiveCreate node.
2. Create a camera using a CameraCreate node.
3. Create a Merge node.
4. Connect the outputs of the PrimitiveCreate, and CameraCreate nodes to the inputs of the Merge node.
5. Create an AttributeScript node.
6. Connect the output of the Merge node to the input of the AttributeScript node.
7. Select the AttributeScript node and press **Alt+E**.

The AttributeScript node becomes editable within the **Parameters** tab.

8. Assign the PrimitiveCreate node as the scene graph location this AttributeScript node is to run on (see [Assigning Locations to a CEL Parameter](#)).

The first steps in the script are the same as in the previous example. You find the translation values of the camera, and the translation values of the primitive, then calculate the vector between them, and its magnitude.

```
from numpy import *
import math

# Set the distance threshold
theThreshold = 1.1;

# Get parameters from the camera
camTrans = GetAttr( 'xform.interactive.translate', atLocation =
'/root/world/cam/camera' );

# Get parameters from the target object
objTrans = GetAttr( 'xform.interactive.translate' );

# Declare a numpy array to hold vector from object to camera
objVec = array( [ 0, 0, 0 ] );

# Get vector from object to camera into objVec
objVec[ 0 ] = objTrans[ 0 ] - camTrans[ 0 ];
objVec[ 1 ] = objTrans[ 1 ] - camTrans[ 1 ];
objVec[ 2 ] = objTrans[ 2 ] - camTrans[ 2 ];

# Get the magnitude of the vector from object to camera
objCamDist = math.sqrt( math.pow( objVec[ 0 ], 2 ) + math.pow( objVec[ 1 ], 2 ) +
math.pow( objVec[ 2 ], 2 ) );
```

Next, find the rotation values of the camera.

```
# Get rotation values from the camera
camTrans = GetAttr( 'xform.interactive.translate', atLocation =
'/root/world/cam/camera' );
camRotX = GetAttr( 'xform.interactive.rotateX', atLocation = '/root/world/cam/camera'
);
camRotY = GetAttr( 'xform.interactive.rotateY', atLocation = '/root/world/cam/camera'
);
camRotZ = GetAttr( 'xform.interactive.rotateY', atLocation = '/root/world/cam/camera'
);
```

Rotation parameters are a four element array for each axis, in the same format used in **.rib** files. The first entry in the array is the magnitude of the rotation, in degrees. The following three entries correspond to the X, Y, and Z axis and show a number between -1 and 1. If the number is zero, then regardless of the magnitude of rotation, no rotation is applied to that axis. If the number is -1 the direction of rotation is reversed. In this example, the values for X, and Z rotation are known to be zero. The magnitude of the Y rotation is found from the first entry in the array returned to camRotY.


```
camRotYMagnitude = camRotY[ 0 ];
```

You need the rotation value in radians for the next step.

```
camRotYRad = math.radians( camRotY[ 0 ] );
```

Create a numpy 3X3 identity matrix, and populate that matrix with values derived from the rotation parameters of the camera.

```
camRotYMatrix = identity( 3, double );
```

```
# Get Sin and Cosine of the camera Y rotation
```

```
camRotYCos = math.cos( camRotYRad );
```

```
camRotYSin = math.sin( camRotYRad );
```

```
camRotYSinMinus = camRotYSin * -1;
```

```
# Fill out the values of the Y rotation matrix
```

```
camRotYMatrix[ 0 ][ 0 ] = camRotYCos;
```

```
camRotYMatrix[ 0 ][ 2 ] = camRotYSin;
```

```
camRotYMatrix[ 2 ][ 0 ] = camRotYSinMinus;
```

```
camRotYMatrix[ 2 ][ 2 ] = camRotYCos;
```

To find the vector from the object to the camera, in the camera frame, multiply the vector from the object to the camera in world frame by the completed Y rotation matrix. First transpose the vector from object to camera to be a column vector, rather than a row vector.

```
objVecT = objVec.transpose();
```

```
# Find objVec adjusted in camera frame
```

```
objVecAdj = dot( camRotYMatrix, objVecT );
```

With the vector from object to camera converted from world frame to camera frame, whether the object is in front of, and too close to the camera is found by examining the value of the Z component of the vector, and calculating the magnitude.

```
# Find the magnitude of objVecAdj
```

```
objVecAdjMag = math.sqrt( math.pow( objVecAdj[ 0 ], 2 ) + math.pow( objVecAdj[ 1 ], 2 )  
+ math.pow( objVecAdj[ 2 ], 2 ) );
```

```
# Set the errors if the object is in front of, and too close to the camera
```

```
if objVecAdjMag < theThreshold and objVecAdj[ 2 ] > 0.0:
```

```
SetAttr( "type", [ "error" ] );
```

```
SetAttr( "errorMessage", [ "Object is too close to the camera" ] );
```

Example Python Scripts

These example scripts assume a basic knowledge of Python and cannot be used to learn the language in isolation.

Sometimes you may get an error if you copy and paste statements from another source, like an e-mail, into the **Python** tab or a parameter. This may be caused by the mark-up or encoding of the source you copied the statement from. To fix the problem, re-enter the statement or correct the indentation manually.

Texture Path Manipulation

If the filename for a texture is included when publishing an asset, the path of where to retrieve that texture needs to be prefixed. If the attribute that contains the filename is `geometry.arbitrary.ColMap` and the path where textures are stored is on the `user.textureRoot` variable, we can write a script to append the two and store the output on the `textures.ColMap` attribute (which is automatically assigned to a parameter of the same name on any shaders, see [Assigning Textures](#)).

1. Create a **PrimitiveCreate** node and add it anywhere in the recipe.
2. Create an **AttributeSet** node and connect it below the **PrimitiveCreate** node.
3. Select the **AttributeSet** node and press **Alt+E**.

The **AttributeSet** node becomes editable within the **Parameters** tab.

4. **Shift**+middle-click and drag from the **PrimitiveCreate** node to the **paths** parameter.

The location created by the **PrimitiveCreate** node is assigned to the **paths** parameter using an expression.

5. In the **attributeName** parameter, enter **geometry.arbitrary.ColMap**.
6. Select **string** from the **attributeType** dropdown.

The **stringValue** parameter displays.

7. Enter a filename in the **stringValue** parameter, for instance **test_file.tx**.



NOTE: The **PrimitiveCreate** and **AttributeSet** nodes are only included to provide sample data. In a normal recipe, the data comes from a published asset which is brought into the scene graph using the **Alembic_In** node (or whatever your studio uses to read in its data). This data would already have the `geometry.arbitrary.ColMap` attribute assigned as part of the asset generation and publication process.

8. Create an **AttributeScript** node and connect it below the **AttributeSet** node.
9. Select the **AttributeScript** node and press **Alt+E**.
The **AttributeScript** node becomes editable within the **Parameters** tab.
10. **Shift**+middle-click and drag from the **PrimitiveCreate** node to the **CEL** parameter.
The location created by the **PrimitiveCreate** node is assigned to the **CEL** parameter.
11. Create a new user parameter called **texturePath**.
12. In the **script** parameter, enter:

```
colMap = GetAttr("geometry.arbitrary.ColMap")

if colMap is not None:
    textureName = user.texturePath[0] + colMap[0]
    SetAttr("textures.ColMap", [textureName])
```

The `GetAttr(<name>)` function returns the value assigned to the attribute `<name>`. In this example, the `GetAttr()` function is returning the value we assigned using the `AttributeSet` node previously in the script. The value returned is always a list (as all attributes in Katana are lists of one particular type, a string in this case).

It is also possible to retrieve attributes at locations in the scene graph (for instance `/root`) and not just the one the script is being run on. An example might be: `GetAttr("renderSettings.cameraName", atLocation="/root")`.



NOTE: It is not possible to return the default value for an attribute using the **GetAttr()** function. Therefore, if the value has not been previously set within the script, the **GetAttr()** function returns `None`.

Also, **GetAttr()** doesn't return inherited values by default, if attribute assigned to `/root/world/geo` that is inherited by `/root/world/geo/robot` is not returned by **GetAttr()** unless you include **inherit=True**, for instance `GetAttr('info.filename', inherit=True)`.

The script starts by assigning the attribute at `geometry.arbitrary.ColMap` to the variable `colMap` using the `AttributeScript` specific function **GetAttr()**. If `colMap` has a value, create a new variable (called `textureName`) from the concatenation of the `user.texturePath` parameter and the `colMap` variable (both of which are lists that need to be referenced by their first element). Finally, set the attribute `textures.ColMap` with the variable `textureName` (once again, set as the first element of a list).

The `textures.ColMap` attribute automatically populates any shaders with a parameter name `ColMap`. For more on textures, see [Assigning Textures](#).

A New Katana Primitive

Everything in Katana is a collection of attributes attached to locations, even the geometry. For example, a polygon primitive in Katana is a location, with attributes detailing individual vertices, and their coordinates. You can create a primitive by creating a scene graph location, and giving that location the attributes required to identify the location as a polygon mesh, and give the coordinates of individual vertices.

In the following example, use a `LocationCreate` node to create the scene graph location, and an `AttributeScript` node to create and set the necessary attributes at that location:

1. Create a `LocationCreate` node and add it to a new recipe.
2. Select the `LocationCreate` node and press **Alt+E**.
The `LocationCreate` node becomes editable within the **Parameters** tab.
3. In the **locations** parameter, enter `/root/world/geo/pyramid`.
4. Create an `AttributeScript` node and connect it below the `LocationCreate` node.
5. Select the `AttributeScript` node and press **Alt+E**.
The `AttributeScript` node becomes editable within the **Parameters** tab.
6. Add the path `/root/world/geo/pyramid` to the **CEL** parameter of the `AttributeScript` node.
7. In the **script** parameter, enter the following:

```
# Set the bounds of the primitive
bounds = [-0.5, 0.0, -0.5, 0.5, 1.0, 0.5]
# Set the coordinates of the 5 vertices to make a pyramid
points = [-0.5, 0.0, -0.5,
-0.5, 0.0, 0.5,
0.5, 0.0, 0.5,
0.5, 0.0, -0.5,
0.0, 1.0, 0.0]
vertexList = [0, 1, 2, 3,
1, 0, 4,
2, 1, 4,
3, 2, 4,
0, 3, 4]
startIndex = [0, 4, 7, 10, 13, 16]

SetAttr("type", ["polymesh"])
SetAttr("bound", ScenegraphAttr.Attr("DoubleAttr", bounds))
SetAttr("geometry.point.P",
ScenegraphAttr.Attr("FloatAttr", points, 3))
SetAttr("geometry.poly.vertexList", vertexList)
SetAttr("geometry.poly.startIndex", startIndex)
```

At this point, you can view the newly created primitive in the **Viewer** tab (although it can't be moved).

To enable the pyramid to be moved, it needs a transformation matrix assigned and a node where changes can be stored. This is handled by the Transform3D node.

1. Create a Transform3D node and connect it to the recipe below the AttributeScript node.
2. Select the Transform3D node and press **Alt+E**.
The Transform3D node becomes editable within the **Parameters** tab.
3. Enter **/root/world/geo/pyramid** in the **path** parameter to associate this Transform3D node with the pyramid's scene graph location.
4. Select **Yes** from the **makeInteractive** parameter dropdown.

The script makes use of a Python function **SetAttr()**. This function is only available inside the AttributeScript node (and not the **Python** tab). It sets the value for an attribute at the current location. The data for scene graph attributes are stored using the ScenegraphAttr.Attr object type.

A More Complex Example

An initializationScript can pass information to the main script, and is typically used where expensive to create data is applied to multiple scene graph locations. Using an initializationScript to pass data to the main script means the data is generated once, rather than once-per-location.

In the following example, you use an initializationScript to:

- Pass information from the initialization script to the main script.
 - Use **GroupBuilder()** to build a group of attributes.
 - Use the ScenegraphAttr.Attr object type to hold the data for the attributes.
1. Create an AttributeScript node and connect it into a recipe after geometry creation.
 2. Select the AttributeScript node and press **Alt+E**.
The AttributeScript node becomes editable within the **Parameters** tab.
 3. Add one or more locations to the **CEL** parameter of the AttributeScript node.
 4. Select **Yes** from the **initializationScript** dropdown.
The **setup** parameter displays.
 5. In the **setup** parameter, enter:

```
import GeoAPI

gb = GeoAPI.Util.GroupBuilder()

scope = ScenegraphAttr.Attr("StringAttr", ["primitive"])
value = ScenegraphAttr.Attr("FloatAttr", [1.0, 0.0, 0.0])
inputType = ScenegraphAttr.Attr("StringAttr", ["color3"])

gb.set("scope", scope)
gb.set("value", value)
gb.set("inputType", inputType)

user.geom = gb.build()
```

6. In the script parameter, enter:

```
SetAttr("geometry.arbitrary.myVariable", user.geom)
```

Any variables that need to be passed from the initialization script to the main script should be prefixed with user (for instance, user.geom in the example above).

The GroupBuilder object is used to assemble a group attribute with other attributes below. As it is possible to use **SetAttr()** with a full explicit hierarchy (without having to first create the sub groups) the GroupBuilder is not usually necessary.



TIP: The initialization script is best used to generate shared data when that shared data is expensive to generate. This example, although valid, would probably be done using the **SetAttr()** function in the main script.

Arbitrary Attributes Within Katana

Katana provides the ability to export arbitrary attributes to renderers. Currently, only attributes defined in **geometry.arbitrary** is written out. This information takes on a renderer-specific form at render time, such as **user data** for Arnold and **primvars** for PRMan.

For each arbitrary attribute to export there are a number of attributes that can be set:

- **scope** - this defines the scope of the attribute. For instance: per object, per face, per vertex. Katana uses its own internal naming for the scope (**primitive**, **face**, **point**, and **vertex**) and then converts them to renderer specific interpretations. For instance, when using PRMan:
 - **primitive** is interpreted as **constant**,
 - **face** is interpreted as **uniform**,
 - **point** is interpreted as **varying**, and
 - **vertex** is interpreted as **face varying**.
- **value** - the actual value for the arbitrary attribute.
- **elementSize** - defines a two-dimensional array by setting the size of each element. For instance, for a list of 10 RGB values (with 30 floats), an **elementSize** of 3 is used.
- **inputType** - specifies the attribute's data type, such as a color, vector, normal.
- **outputType** - converts the input data type into a different type for output.
- **indexedValue** - in conjunction with **index**, defines an indexed array.
- **index** - the indices of the array of values stored in **indexedValue**.

Some Example Arbitrary Attributes

This is the simplest form an arbitrary attribute can have, a scope and a value.

```
SetAttr("geometry.arbitrary.myFloat.scope", ["primitive"])
SetAttr("geometry.arbitrary.myFloat.value", [1.0])
```

Here, a single float is converted to a color3 data type.

```
SetAttr("geometry.arbitrary.myFloatToColor3.scope",
["primitive"])
SetAttr("geometry.arbitrary.myFloatToColor3.value", [0.1])
```

```
SetAttr("geometry.arbitrary.myFloatToColor3.outputType",
["color3"])
```

In this example, two colors are defined and then converted to uniform color[2] when rendering with PRMan.

```
SetAttr("geometry.arbitrary.myColorArray.scope",
["primitive"])
SetAttr("geometry.arbitrary.myColorArray.value",
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
SetAttr("geometry.arbitrary.myColorArray.inputType",
["color3"])
SetAttr("geometry.arbitrary.myColorArray.elementSize", [3])
```

A simple example with each face of a cube assigned an alternating texture using an indexed array.

```
SetAttr("geometry.arbitrary.myIndexedArray.scope",
["face"])
SetAttr("geometry.arbitrary.myIndexedArray.indexedValue",
["textureA.tx", "textureB.tx"])
SetAttr("geometry.arbitrary.myIndexedArray.index",
[0, 1, 0, 1, 0, 1])
```

Beyond the AttributeSet and AttributeScript Nodes

Using the AttributeScript node does have its limitations, most notably speed. Katana provides a C++ API that allows you to manipulate attributes using a plug-in, called Attribute Modifier Plug-in (AMP). The *API documentation* can be found through the **Help > API Reference > Plug-in API menu**.

An example AMP implementation is included with Katana that enables you to define attributes in XML and assign them to scene graph locations. The example node is AttributeFile_In and the *Katana Technical Guide* that accompanies the node can be found in the **Help > Documentation** menu.

The FnAttribute Module

The **FnAttribute** module provides the implementation of *attributes* - the fundamental data storage types used for Katana's scene graph. **FnAttribute** is provided to any C++, Python, or Lua plug-in, including Ops, OpScripts, and AttributeScripts. Essentially, any plug-in that queries, provides, or manipulates scene data makes use of this module.

Data Attributes

Katana's basic *data attributes* can be of integer (signed, 32-bit), float, double, or string type, and are represented by the **IntAttribute**, **FloatAttribute**, **DoubleAttribute**, and **StringAttribute** classes.

Each data attribute contains a map of time samples, where each sample is a contiguous array of the base data type. Every data attribute is therefore both multi-valued (even if there is only one value) and multi-sampled (even if there is only one sample). Note that all samples must contain the same number of values, and that it is legal, albeit unusual, for a data attribute to contain no samples and/or no values.

Katana interprets an attribute's sample times as *time in frames relative to the current frame*; that is, $t=0.0$ indicates the time for the current frame and $t=0.5$ indicates a time 0.5 frames ahead of the current frame.

Finally, Katana's data attributes have the notion of a *tuple size*, which allows you to partition the values of a sample into equal-size groups. Unlike the *value count* or the *sample count*, the tuple size has no effect on how the actual data that is stored, and is used purely as a hint for how the attribute should be represented in Katana's UI.

Group Attributes

In addition to these data attributes, **FnAttribute** also provides *group attributes* (represented by the **GroupAttribute** class). Group attributes contain an **ordered** mapping of names to other attributes. The mapped type can itself be a group attribute, supporting the creation of arbitrarily-nested attribute structures. Group attributes support lookups by both index and name, but note that group attributes are not a general-purpose dictionary structure, and that look-ups by name are less efficient.

Null Attributes

A special type of attribute is the *null* attribute, represented by the **NullAttribute** class. A null attribute is an attribute with **no** value. Its meaning depends on the context in which it is used. One use of **NullAttribute** in Katana is to prevent the inheritance of particular attributes of a location at its child locations. See [Group Inheritance and the groupInherit Flag](#) for more information.

Mutability

An important property of Katana's attributes is that they are *immutable* - an attribute cannot be modified after creation. This allows efficient sharing of an attribute's underlying data by multiple consumers, including those living on different threads. However, it does imply that *updating* an attribute takes more work: you must copy the data out of the attribute into some mutable data structure, make the changes as needed, then create a brand new attribute using the contents of this data structure.

Accessing the FnAttribute Module

C++

```
#include <FnAttribute/FnAttribute.h>
```

Members of the **FnAttribute** module are located under the **FnAttribute** namespace.

Python

In AttributeScripts, the **FnAttribute** module exists in the global namespace, and no explicit import is necessary. From other Python plug-in contexts, it can be imported from the global **Katana** module, that is:

```
from Katana import FnAttribute
```

Lua (OpScript)

The members of the **FnAttribute** module are made available in the global namespace; no import is necessary.

Working with Data Attributes



NOTE: For brevity, the C++ and Python code snippets that follow assume the members of the **FnAttribute** module have been imported into the global namespace, and omit the **FnAttribute::** or **FnAttribute.** prefix.

Creating an Attribute

Listed below are examples of constructing attributes that contain:

- a single sample and value
- multiple values
- multiple values and multiple samples

C++

```
// Single value in a single sample at t=0.0.
IntAttribute myAttr(42);

// Four values in a single sample at t=0.0.
float values[] = {0.1f, 0.2f, 0.3f, 0.4f};
FloatAttribute myMultiValuedAttr(values, 4 /*valueCount*/, 1 /*tupleSize*/);

// Four values in a single sample at t=0.0, explicit tuple size of 4.
float values[] = {0.1f, 0.2f, 0.3f, 0.4f};
FloatAttribute myMultiValuedAttrExplicitTupleSize(
    values, 4 /*valueCount*/, 4 /*tupleSize*/);

// Two samples (at t=0.0 and t=1.0), each with four values.
float sampleTimes[] = {0.0f, 1.0f};
```

```
float values[] = {
    0.1f, 0.2f, 0.3f, 0.4f,
    0.5f, 0.6f, 0.7f, 0.8f,
};
const float* sampleValues[] = {&values[0], &values[4]};
FloatAttribute myMultiSampledAttr(
    sampleTimes, 2 /*sampleCount*/,
    sampleValues, 4 /*valueCount*/,
    1 /*tupleSize*/);
```

Python

```
# Single value in a single sample at t=0.0.
myAttr = IntAttribute(42)

# Four values in a single sample at t=0.0.
myMultiValuedAttr = FloatAttribute([0.1, 0.2, 0.3, 0.4])

# Four values in a single sample at t=0.0, explicit tuple size of 4.
myMultiValuedAttrExplicitTupleSize = FloatAttribute([0.1, 0.2, 0.3, 0.4], 4)

# Two samples at t=0.0 and t=1.0, each with four values.
myMultiSampledAttr = FloatAttribute({
    0.0: [0.1, 0.2, 0.3, 0.4],
    1.0: [0.5, 0.6, 0.7, 0.8],
})
```

Lua (OpScript)

```
-- Single value in a single sample at t=0.0.
local myAttr = IntAttribute(42)

-- Four values in a single sample at t=0.0.
local myMultiValuedAttr = FloatAttribute({0.1, 0.2, 0.3, 0.4})

-- Four values in a single sample at t=0.0, explicit tuple size of 4.
local myMultiValuedAttrExplicitTupleSize = FloatAttribute(
    {0.1, 0.2, 0.3, 0.4}, 4)

-- Two samples at t=0.0 and t=1.0, each with four values.
local myMultiSampledAttr = FloatAttribute({
    [0.0] = {0.1, 0.2, 0.3, 0.4},
    [1.0] = {0.5, 0.6, 0.7, 0.8},
})
```

```
})
```

Reading the Values of a Sample

The following examples assume **myFloatAttr** is a valid **FloatAttribute**.

C++

```
// |myFloatValues|, as returned by getNearestSample(), is little more than a
// pointer to the underlying data. This pointer will become dangling if
// `myFloatAttr` goes out of scope, so beware!
IntAttribute::array_type myFloatValues = myFloatAttr.getNearestSample(0.0f);
for (IntAttribute::array_type::const_iterator it = myFloatValues.begin();
     it != myFloatValues.end() ++it) {
    // ...
}

// In C++, you can also access the underlying data directly.
const float* underlyingSampleBuffer = myFloatValues.data();
```

Python

```
# In Python, getNearestSample() returns a copy of the underlying sample
# data as a Python list object.
myFloatValues = myFloatAttr.getNearestSample(0.0)
for value in myFloatValues:
    # ...
```

Lua (OpScript)

```
-- In Lua, getNearestSample() returns a copy of the underlying sample data,
-- and returns it as a Lua table. As a Lua table, myFloatValues' indices
-- begin at 1, not 0.
local myFloatValues = myFloatAttr:getNearestSample(0.0)
for idx, value in ipairs(myFloatValues) do
    -- ...
end
```

Reading a Single Value

A common case is for an attribute to contain a single value. The following examples demonstrate how to extract this value. We return a default value of 0.0 if the attribute is not a float attribute, or has no values.

C++

```
float myValue = FloatAttribute(myAttribute).getValue(
    0.0, false /*throwOnError*/);
```

Python

```
if isinstance(myAttribute, FloatAttribute):
    myValue = myAttribute.getValue(0.0, False)
else:
    myValue = 0.0
```

Lua (OpScript)

```
local myValue = Attribute.GetFloatValue(myAttribute, 0.0)
```

Working with Group Attributes

GroupAttribute instances are generally created indirectly using the **GroupBuilder** helper class.

To create a new group attribute, first instantiate a **GroupBuilder**. You may then mutate the builder using various operations, for example adding (**name, attribute**) pairs to it using the **set()** method, or deleting an existing attribute by passing its name to **del()**. When done, retrieve the newly constructed **GroupAttribute** with the **GroupBuilder**'s **build()** method.

It's also possible to update a **GroupBuilder** with the contents of an existing group attribute using the **update()** and **deepUpdate()** methods. The **update()** method performs a *shallow* merge, where existing attributes are overwritten if they share the same name as the new attributes. A common pattern is to call **update()** on an empty builder to pre-populate it with the contents of an existing group attribute. The **deepUpdate()** method is the *recursive* version of **update()**, effectively overlaying the contents of the incoming groups atop the existing groups of the builder.

As a convenience, **GroupBuilder** supports creating arbitrarily nested attribute structures by passing a dot-delimited path to its **set()** and **del()** methods. (This implies that the . (period) character is not a valid attribute name!)



NOTE: Unlike the types mentioned so far, **GroupBuilder** is not itself an **Attribute**.

The code snippets listed below show the creation of a group attribute with the following structure:

```
{
  "my": {
    "nested": {
      "attribute": IntAttribute(2)
    }
  },
  "myTopLevelAttribute": StringAttribute("taco"),
```

```
"myOtherTopLevelAttribute": FloatAttribute(4.0f)
}
```

C++

```
GroupBuilder gb;
gb.set("my.nested.attribute", IntAttribute(2));
gb.set("myTopLevelAttribute", StringAttribute("taco"));
gb.set("myOtherTopLevelAttribute", FloatAttribute(4.0f));

GroupAttribute groupAttribute = gb.build();
// |groupAttribute| now has the structure listed above; |gb| is empty.
```

Python

```
gb = GroupBuilder()
gb.set("my.nested.attribute", IntAttribute(2))
gb.set("myTopLevelAttribute", StringAttribute("taco"))
gb.set("myOtherTopLevelAttribute", FloatAttribute(4.0))

groupAttribute = gb.build()
# |groupAttribute| now has the structure listed above; |gb| is empty.
```

Lua (OpScript)

```
local gb = GroupBuilder()
gb:set("my.nested.attribute", IntAttribute(2))
gb:set("myTopLevelAttribute", StringAttribute("taco"))
gb:set("myOtherTopLevelAttribute", FloatAttribute(4.0))

local groupAttribute = gb:build()
-- |groupAttribute| now has the structure listed above; |gb| is empty.
```

Notes

There is no way to inspect the contents of the builder while you are mutating it. Instead, you must call **build()** and inspect the generated **GroupAttribute**. Note that, by default, calling **build()** clears the contents of the builder, and to override this behavior you must pass the constant **GroupBuilder::BuildAndRetain** (C++), **GroupBuilderBuildAndRetain** (Python), or **GroupBuilder.BuilderBuildMode.BuildAndRetain** (Lua) to **build()**.

Note on Backwards Compatibility

Previous versions of Katana would retain the contents of the builder when calling **build()**. Customers with existing C++ plug-ins they wish to use in Katana 2.0v1, and after, are advised to audit their uses of **GroupBuilder::build()** to

ensure the new semantics do not cause unintended side effects.

Group Inheritance and the `groupInherit` Flag

Group attributes have a special flag called `groupInherit`. Setting this flag to **True** signals to Katana that all attributes contained in the group should be inherited by child locations.

By default, group attributes created by `GroupBuilder` have their `groupInherit` flag set to true. To disable this, call **`setGroupInherit(false)`** (C++, Lua) or **`setGroupInherit(False)`** (Python) on the builder. This `groupInherit` flag on the builder is "sticky": **`setGroupInherit()`** may only be called once per builder, and subsequent calls have no effect.

As the name suggests, group inheritance is a property of the *group* and not the data attributes contained within it. The use of **`NullAttribute`** in combination with group inheritance allows for fine-grained control over which attributes are inherited by child locations. Specifically, Katana interprets a null attribute in a group as a signal to ignore inherited values for the attribute with the given name, forcing the default value if available. (If no default value is available the attribute is simply deleted.)

For more information about attribute inheritance, see *Inheritance Rules for Attributes* in the *Katana Technical Guide*.

Invalid Attributes (C++ only)

In the C++ variant of the **`FnAttribute`** module, there is the concept of an *invalid Attribute*, which is akin to the **`NULL`** pointer in C, **`None`** in Python, or **`nil`** in Lua. When an attribute instance is invalid, its **`isValid()`** method returns **`false`**.



NOTE: This is distinct from the **`NullAttribute`** attribute type. The **`NullAttribute`** represents an Attribute with no value; invalid attributes represent the *not-an-attribute* value. A **`NullAttribute`** is still a valid attribute!

Sources of invalid attributes include:

- Default-constructed attributes:

```
IntAttribute intAttr;
// intAttr.isValid() == false
```

- Invalid type conversions: Attribute types are strictly enforced. There is no implicit conversion from, say, a **`FloatAttribute`** to a **`DoubleAttribute`**, whereas in C, a **`float`** is implicitly convertible to a **`double`**.

```
DoubleAttribute doubleAttr = myGroupAttr.getChildByName(
    "someFloatAttribute");
// doubleAttr.isValid() == false, because the attribute named
// "someFloatAttribute" in myGroupAttr was of type FloatAttribute, not
// DoubleAttribute
```

- Sentinel return value indicating the requested child doesn't exist::

```
Attribute childAttr = myGroupAttr.getChildByName(
```

```
"someNonExistentChild");
// childAttr.isValid() == false
```

While **intAttr**, **doubleAttr** and **childAttr** are considered *invalid*, it is still legal to call their methods, and *sensible* values - generally 0 - are returned. (An exception to this is the **getValue()** method, which by default throws an exception if the attribute is invalid.)

Note that invalid attributes have no analogue in Python or Lua, where **None** or **nil** are used to represent the *attribute not found* case.

Error Checking

Writing robust code requires that you do some sanity-checking when retrieving an **Attribute** from an unknown source - for example, from the scene using the Geolib3 cook interface - to be sure that the attribute conforms to what your code is prepared to deal with.

Two common sources of error are assuming an attribute is *valid* (C++), not **None** (Python), or **nil** (Lua), or not validating that its type is what you expect.

C++

In C++, a standard idiom is to cast a suspicious attribute to the desired type, and call **isValid()** to see if the cast succeeded.

```
StringAttribute stringAttr = interface.getAttr("mayOrMayNotExist");
if (!stringAttr.isValid())
{
    throw std::runtime_error("mayOrMayNotExist does not exist "
                             "or is not a string");
}
// At this point, we know it's a valid string attribute, so we're happy.
```

Python

In Python, you can use the built-in **isinstance()** function to check if a given object is an attribute of a given type.

```
stringAttr = GetAttr("mayOrMayNotExist", asAttr=True, asFnAttribute=True)
if not isinstance(stringAttr, StringAttribute):
    raise TypeError("mayOrMayNotExist does not exist or is not a string")
# At this point, we know it's a valid string attribute, so we're happy.
```

Lua (OpScript)

In Lua, there is a set of utility functions on the Attribute module for verifying that some object is an attribute of a given type.

```

local stringAttr = Interface.GetAttr("mayOrMayNotExist")
if Attribute.IsString(stringAttr) then
    error("mayOrMayNotExist does not exist or is not a string")
end
-- At this point, we know it's a valid string attribute, so we're happy.

```

Debugging

A useful debugging aid for inspecting attribute data is to use the **getXML()** method to print out an XML representation of an attribute. Note that this is less effective when attributes contain a large number of values or samples, as the resulting XML string can be very large.

C++

```

IntAttribute myAttr(42);
std::cout << myAttr.getXML();

```

Python

```

myAttr = IntAttribute(42)
print(myAttr.getXML())

```

Lua (OpScript)

```

local myAttr = IntAttribute(42)
print(myAttr:getXML())

```

Each of the above prints the following to the standard output stream:

```

<attr tupleSize="1" type="IntAttr">
  <sample size="1" time="0" value="42" />
</attr>

```


Look Files

Overview

Katana's Look Files are a powerful general purpose tool that can be used in a number of ways. In essence they contain a baked cache of changes that can be applied to a section of scene graph to take it from an initial state to a new one.

Typically they are used to store all the relevant changes that need to be applied to take an asset from its raw state, as delivered from modeling with no materials, to a look developed state ready for rendering. They can also be used for other purposes such as to contain palettes of materials or to hold show standard settings for render outputs such as image resolutions, anti-aliasing settings and what output channels (AOVs) to use.

Different studios define the tasks done by look development and lighting in different ways. In this section we're going to look at what could be considered a typical example of the tasks to give a clear example of possible use, but the actual work done by different departments could be different. Look files should be seen as a useful flexible general tool that can be used to pass baked caches of scene graph modifications from one KATANA project to another.

Handing off Looks from Look Development to Lighting

The most standard use of Katana's Look Files is to describe what materials are needed for a given asset, such as a character, car or building, and which material is assigned to each geometry location. The Look File can also record any overrides such as modifications to shaders on particular locations, for example if a given object needs the specular intensity on a shader setting to a special value. The Look File can also record the shaders and assignments that are needed for a number of different passes, such as if you are going to do separate renders for passes such as the beauty pass, ambient occlusion, volumetric renderer.

The traditional workflow is that Look Development defines how each asset should look in all the different render passes required. They then 'bake' out a Look File for each asset, or multiple Look Files if there are a number of alternative look variants for an asset.

The Look File records this data in a form that can be re-applied to the 'naked' asset in Lighting. In Lighting the appropriate Look File is assigned to each asset. Downstream in the Katana graph when you want to split into all the different separate passes you do a 'LookFileResolve' which actually does the work of re-applying all the materials and other changes to the asset that are needed for a given pass.

Look File Baking

Look Files are written out by using the LookFileBake node. Using this node you have to set one input to a point in the node graph where the scene data is in its original state and another to indicate the point in the node graph where the scene data is in its modified state. If you want to include multiple output passes in the Look File you can add additional inputs to connect to points in the node graph where the scene data has been set up for that extra pass.

During LookFileBake every location in the scene graph under the root location is compared with the equivalent scene graph location in the original state. What is written out into the Look File are all the changes, such as changes to attributes (new attributes, modified values of existing attributes, and any attributes that have been deleted).

The details of any new locations that have been added are also written out. This means that new locations that are part of the 'look' can be included, such as face-sets for a polygon mesh that weren't part of the original model, or to add lights such as architectural lights on a building.

One important thing to note here is that while the nodes in the Node Graph represent live recipe, the Look File is a baked cache of the results of those nodes: it's a list of all the changes that the nodes make to the scene graph data rather than the recipe itself.

One of the main reasons for using Look Files rather than keeping everything as live recipe is efficiency. If you have thousands of assets, like you could in a typical shot from a CG Feature or VFX film, it can be inefficient to keep everything as live recipe. The Look Files allow the changes needed to be calculated once and then recorded as a baked list by comparing the state of the scene graph data before and after the filters. If you want to make additional changes in lighting on top of those defined by a Look File you still can do so by using additional overrides.

If a new version of the asset is created, any associated Look Files need to be baked out again by re-running the LookFileBake in the appropriate Katana project.

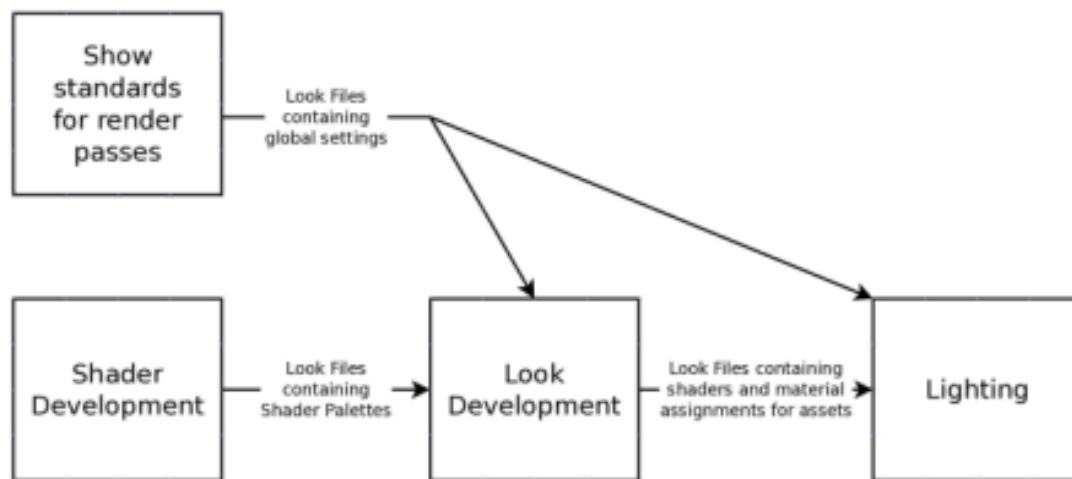
Conversely, if you want to hand off live recipe from one Katana project to another one you should use macros or LiveGroups instead.

Other Uses of Look Files

As mentioned previously, Look Files are actually quite a flexible tool that can be used for a number of different purposes as well as their 'classic' use to hand off looks for Look Dev to Lighting. Some of the other things they can be used for include:

- Defining palettes of standard shaders to use in a show.

- Making additional modifications to assets beyond simple shader assignment, such as:
 - Visibility settings to hide objects if they shouldn't be visible.
 - Adding face-sets to objects for per-face shader assignment.
 - Per-object render settings, such as renderer specific tessellation settings.
 - Defining additional lights that need to be associated with an asset, such as a car that needs head lights or a building that needs architectural lights.
 - Adding additional locations to the asset such as new locations in the asset hierarchy for hair procedurals.
- Specifying global settings for render passes, such as what resolution to render at, defining what outputs (AOVs) are available and anti-aliasing settings.



How Look Files Work

To gain a better understanding of what Look Files are and how they can be used we are going to look in more detail at how they actually work.

The geek-eye view of a Look File is that it's a 'scene graph diff'. In other words it's a list of all the changes that need to be made to a sub-branch of the scene graph hierarchy to take from an initial state (typically a model without any materials assigned) to a new transformed state (typically the model with all its materials assigned to correct pieces of geometry, and any additional overrides such as shader values or to change visibility flags). When you do a `LookFileResolve` all those changes are re-played back onto the relevant scene graph locations.

To make material assignments work all the materials assigned within the hierarchy are also written out into the Look File. Similarly, any renderer procedurals required are also written out into the Look File.

For each render pass a separate scene graph diff is supplied. There are two caveats we should mention about Look Files:

- The data in Look Files is non-animating, so you can't use them to hand off animating data such as flashing disco lights or lightning strikes. Animating data like this can be handled in a number of ways, including making the

animating data part of the 'naked' asset, or by using Katana Macros and Live Groups to hand off actual Katana node that can have animating parameters.

- Currently you can't delete scene graph locations using Look Files, you can only add new locations or modify existing ones. For instance, to hide an object you should set its visibility options rather than pruning it from the scene graph.

Setting Material Overrides using Look Files

Following the core principle in Katana that all scene data should be inspectable and modifiable, mechanisms are needed to allow material settings defined in Look Files to be overridable downstream.

In Katana this is done by bringing the materials into the scene so that the user can use the normal Katana nodes, such as the Material node in 'override' mode, so make changes.

For efficiency, and to avoid lighters scenes becoming littered with every single material that may be used on any asset in the scene, the materials used in by a Look File aren't loaded into scenes by default. If you want to set overrides on the materials you first need to use a `LookFileOverrideEnable` node. This brings in the materials from the Look File into the Katana scene and sets them up (by bringing them in a specific locations in the scene graph hierarchy based on the name of the Look File) so that these instances are used instead of the ones contained in the original Look File.

`LookFileOverrideEnable` also brings in any renderer procedurals for overriding in a similar manner to materials.

Collections using Look Files

Look Files can also be used to pass off Collections from one Katana project to another.

When a Look File is baked out for a sub-section of the scene graph hierarchy, for every Collection the baking process notes if any locations inside that sub-section of the hierarchy are in the Collection. If they do the paths for those matching locations are written into the Look File.

When the Look File is brought in and resolved, these baked sub-collections are brought in as Collections that are available at the root location of the asset.

In essence this means that if you're using a Look File to pass off the materials and assignments on an asset from Look Dev to Lighting you can also declare Collections in your Look Dev scene so that they are conveniently available to the lighters.

Look Files for Palettes of Materials

As well as being used to contain the Materials used by a specific asset, Look Files can also be used to contain general collections of shaders. This is particularly useful if you want to have studio or show standard sets of shaders created in 'shader development' which are then made available to other users. Another use of shaders from Look Files is to pre-define standard shader sets for lights (including light shaders made out of network shaders) to be brought in for Lighting.

Materials can be written out into a Look File using the LookFileMaterialsOut node. LookFileBake also has an option to 'alwaysIncludeSelectedMaterialTrees' that allows the user to specify additional locations of materials they want to write out into the Look File whether or not they are assigned to geometry.

To bring the materials from a Look Files into a project you can use the LookFileMaterialsIn node.

Look File Globals

Look Files can also be used to store global settings, so that these values can be brought back in as part of the Look File Resolve. This is usually used to define show standard settings for different passes. It can be used to set things such as:

- What renderer to use for a given pass
- What resolution and anti-aliasing settings to use
- Any additional render output channels (AOVs) you want to define and use.

When using LookFileBake you can specify the option to 'includeGlobalAttributes'. Enabling this option means that any values set at **/root** are stored in the Look File.

To specify which Look File to use to set global settings use the LookFileGlobalsAssign node.

Lights and Constraints in Look Files

If lights and constraints are declared in a Look File there is some special handling needed when they are brought in because knowledge of lights and constraints is generally needed at the global scene level.

There is a special node called LookFileLightsAndConstraintsActivator designed to declare any Look Files that are being used that declare lights and constraints. This handles the work of adding them to the global lists held at **/root/world**.

The Look File Manager

The LookFileManager is provided to simplify the process of resolving Look Files, applying global settings, and allow the users to specify overrides such as for materials. This is a Super Tool that makes use of many of the more atomic operation nodes mentioned previously such as LookFileResolve, LookFileOverrideEnable and LookFileGlobalsAssign.

In particular it is designed to help make setting material overrides that need to be applied to some passes but not all passes a lot easier for the user.

Look Development with Look Files

Look File Overview

The primary use for look files is to store the changes from one state of the scene graph to another. This is how a look development artist records the changes from a bare asset to its completed state. Other departments can use look files to record:

- the renderer settings for a show (recorded at **/root**), such as the renderer, resolution and what AOVs to output.
- a material palette, used either within the look development department or later during lighting.

Using Look Files to Create a Material Palette

Look files can be used to create a material palette. This material palette can be brought into other recipes, allowing material presets to be setup and shared across assets, shots, and scenes. A material palette can also be passed to the lighting department with typical light materials to be assigned to lights, for instance, using the GafferThree node.

Creating a Material Palette

The LookFileMaterialsOut node writes all materials at or below the location **/root/materials** to a Katana look file. This look file is designed to be a material palette that can then be read in by those in look development to help design an asset's look but can also be used in lighting if the materials are light shaders.

To create a material palette:

1. Create the materials for the material palette. For information on the creation of materials, see [Adding and Assigning Materials](#).
2. Create a LookFileMaterialsOut node and connect it to the bottom of the recipe.
3. Select the LookFileMaterialsOut node and press **Alt+E**.
The LookFileMaterialsOut node becomes editable within the **Parameters** tab.
4. Enter the location for the Katana look file (**.klf**) in the **saveTo** parameter.
5. Click **Write Look File**.

The **Save Materials to Look File** dialog displays.

6. Confirm the location of the Katana look file within the dialog and click **Accept**.

The look file is saved.

Reading in a Material Palette

Once you've created a material palette, it can then be easily added to any asset's look development recipe.

To read in a material palette:

1. Create a **LookFileMaterialsIn** node and connect it to the recipe. It is usually added in a separate branch and joined with a **Merge** node.
2. Select the **LookFileMaterialsIn** node and press **Alt+E**.

The **LookFileMaterialsIn** node becomes editable within the **Parameters** tab.

3. Enter the location for the material palette's Katana look file (**.klf**) in the **lookfile** parameter.
4. Select the pass from the Katana look file to use for this palette with the **passName** parameter.
5. Select whether or not to bring in the materials palette by reference using the **asReference** dropdown.

When reading the material palette by reference, any materials assigned keep a reference to the Katana look file from which they got their material. Thus, if the material in the materials palette Katana look file is updated, so is the material assigned to the asset. This happens even if the asset's look development is saved in a new Katana look file. If by reference is not used, the asset's look development Katana look file is baked and not updated.

6. Using the **locationForMaterials** dropdown, select where in the scene graph to import the materials from:
 - **Load at original location** - the materials maintain the same location.
 - **Load at specified location** - provides a parameter, **userLocation**, that acts as a namespace for the material palette. For instance, a material at **/root/materials/geo/chrome** with **userLocation** default_pass is placed at **/root/materials/lookfile/default_pass/geo/chrome**.

If a location already exists, it is overwritten.

Using Look Files in an Asset's Look Development

Katana look files (**.klf**) can be used for an asset's look development. They are created by comparing the scene graph generated at two points within the Node Graph and then recording the difference. When that same asset is used within another recipe, the look file can be applied, restoring the state created during look development. Multiple looks (within the same file) can be created for different passes, the first pass is always called **default**.

Creating a Look File Using LookFileBake

The **LookFileBake** node is used to compare the scene graph generated at two points within the node graph, an original and a second point downstream of the original. At each location below the **LookFileBake** node's

rootLocations parameter the difference between the original scene graph and the downstream scene graph is recorded.

Creating a Look File

1. Create a LookFileBake node and place it anywhere within the **Node Graph**.
2. Connect from a point in the recipe where the bake **asset** has no materials assigned to the **orig** input of the **LookFileBake** node.



TIP: Connecting from a point straight after the geometry has been imported usually produces the best results.

3. Connect an output from downstream in the recipe, where the asset has the look you want to bake, to the **default** input of the LookFileBake node.
4. Select the LookFileBake node and press **Alt+E**.
The LookFileBake node becomes editable within the **Parameters** tab.
5. In **rootLocations**, enter the scene graph location to traverse.



TIP: It is a good idea to make sure **rootLocations** matches the location the asset was initially imported. You can traverse multiple locations by using **Add Locations** to the right of the **rootLocations** parameter. For more information on adding path locations using location parameters, see [Changing a Node's Parameters](#).

6. Enter the asset name for the look file in the **saveTo** parameter.
7. Click the **Write Look File** button.
The **Write Look File** dialog displays.
8. Select where the asset is going to be saved (it defaults to the **saveTo** parameter) and click **Accept**.
Katana starts to bake out the look file. This may take some time as all locations in **rootLocations** must be fully expanded for each pass and all their attributes compared. Any differences detected between the scene graph generated at the **orig** input and the scene graph generated at the pass inputs are written to the look file.

Adding Additional Locations, and Using rootIds

A LookFileBake can compare multiple original points (root locations) with a single downstream location, potentially recording the changes to multiple assets, located under different scene graph branches. When resolving a Look File with multiple root locations, Katana attempts to match root locations in the Look File, with scene graph locations in the target scene.

You can specify a user attribute called **rootId** for any scene graph location, and - if that location is used as a root locations in a LookFileBake - use rootIds to help determine which scene graph location the resulting Look File is resolved to. A rootId is an attribute of type string, under **lookfile.rootId** in a scene graph location's Attributes.



NOTE: To create a user attribute **lookfile.rootId**, use an AttributeSet node pointed to the target location. Set the **action** field to **Create/Override**, the **attributeName** field to **lookfile.rootId**, the **attributeType** to **string**, and **groupInherit** to **Yes**. In the **stringValue** field, enter your chosen rootId. For more on using AttributeSet nodes, see [Making Changes with the AttributeSet Node](#).



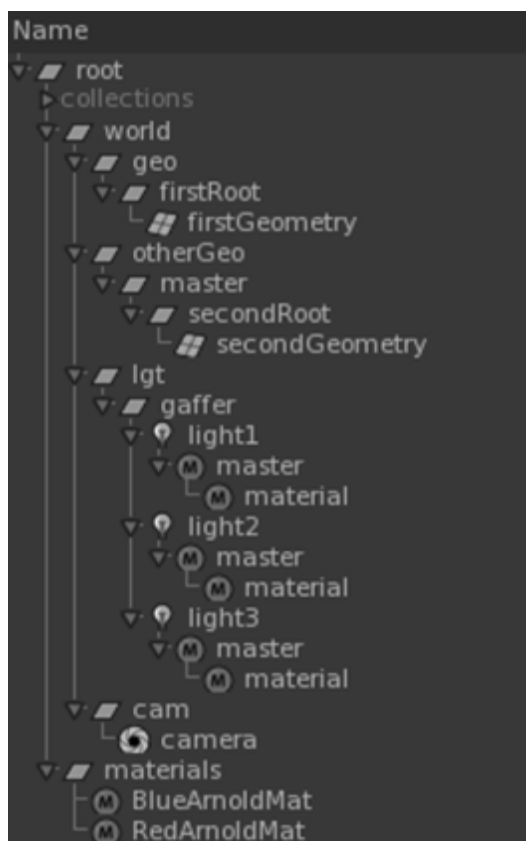
NOTE: You can select any scene graph location as a root location. It determines the first level of the relative path - or paths - generated by a LookFileBake.

When a LookFileAssign is resolved, local paths are determined using either the root location names of source and target, or if specified by the unique rootIds of source and target root locations. Materials from a Look File are applied using a combination of the determined local paths, and the name of the location the material is applied to.

In an example with multiple root locations, and materials at multiple child locations, there are a number of possible outcomes when the resulting LookFile is resolved:

- With rootIds set, and matching location names, the materials are assigned as expected.
- With no rootIds, but location names the same, one of the multiple source materials is assigned. There's no way to guarantee which one.
- With no rootIds and different location names, one of the multiple source materials is assigned. Again there's no way to guarantee which one.

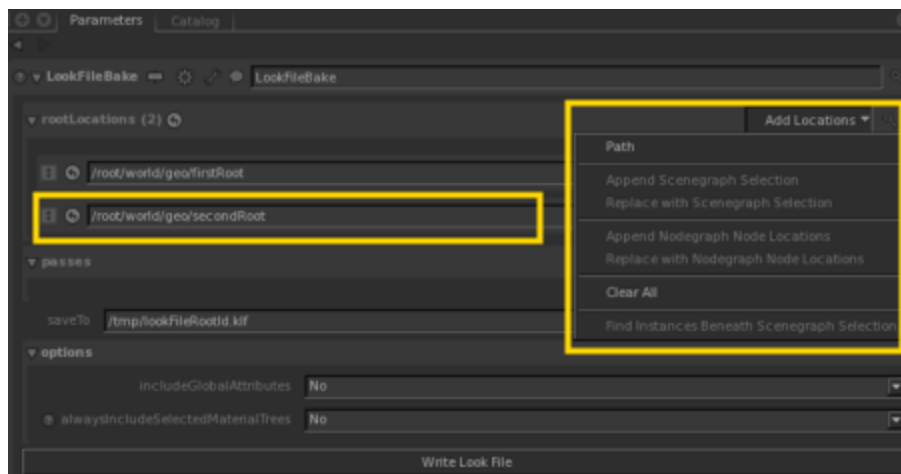
The scene graph shown below has multiple root locations **firstRoot** and **secondRoot**, with geometry under each.



In this case, each root location has geometry with applied materials in the path below it, and we want to include those materials in a Look File. We also want to use rootIds to determine the resolve locations of the Look File, so scene graph locations **firstRoot** and **secondRoot** each have a unique rootId. See [Look File Overview](#) for more on Look Files, and their uses.

To use rootIds in a LookFileBake with multiple root locations, first set rootIds on the root locations, then add each root location to the **rootLocations** field in a LookFileBake node before writing a LookFile:

- In a scene like the one shown below, with multiple scene graph root locations, add rootIds to each root location using an AttributeSet node.
- Add a LookFileBake node.
- Edit the LookFileBake node, select **Add Locations > Path**, in the node's **Parameters** tab and enter the path of a scene graph location you want to add into the resulting **rootLocations** field. Repeat for each additional root location you want to add.



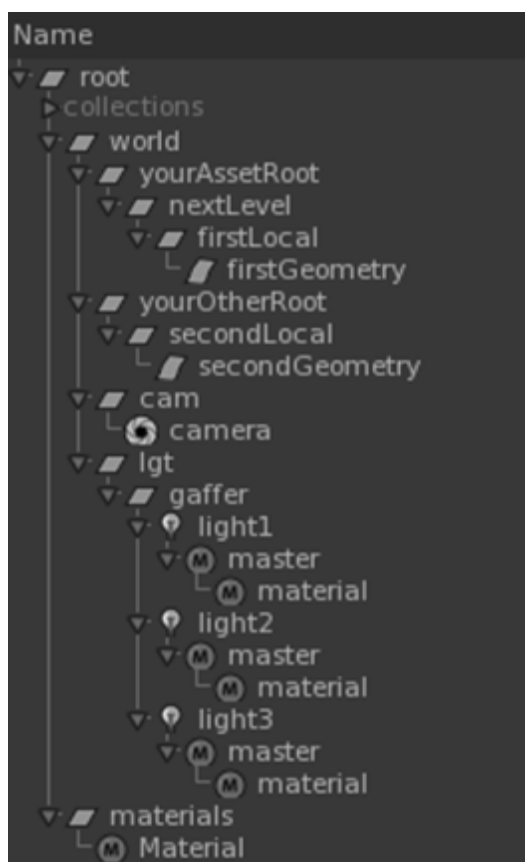
- Write the Look File.

The Look File stores local paths using each rootId as the top level, as well as the asset names.



NOTE: If the chosen root location has a **rootId**, it is included in the LookFileBake. If not, the location name is used.

Using a LookFileAssign node, bring the Look File into a new scene, with different scene graph location names, and paths (such as the scene shown below).



Although the paths, and path names are different, the geometry locations **firstGeometry** and **secondGeometry** have the same relationship to locations **firstLocal** and **secondLocal** that the same locations did to **firstRoot** and **secondRoot** in the original scene. So, as long as **firstLocal** shares a rootId with **firstRoot**, and **secondLocal** with **secondRoot**, and the geometry location names are the same, the Look File assigns as expected.

Adding Additional Passes to a Look File

1. In the **LookFileBake** node, select **Add > Add Pass Input** to the right of the **passes** parameter grouping.
A new pass **name** parameter displays.
2. Type the name of the new pass in the **name** parameter.
A new input is added to the **LookFileBake** node, named according to the name of the new pass
3. Connect the new input of the **LookFileBake** node to the output of the point in the recipe you want to record the look of.

Having the Look File Include any Changes to /root

Select **Yes** for the **includeGlobalAttributes** dropdown inside the **options** parameter grouping of the LookFileBake node.

Including Materials within the Look File

Look files automatically include materials that are assigned to geometry below locations it traverses (as are renderer procedurals). On occasion it might be useful to include extra materials created during look development to be read in later using the LookFileMaterialsIn or Material nodes.

To force materials to be included within the look file:

1. In the **options** parameter grouping of the LookFileBake node, select **Yes** for the **alwaysIncludeSelectedMaterialTrees** dropdown.
A locations widget displays.
2. In **selectedMaterialTreeRootLocations**, enter the material root scene graph location of the materials to include.
Multiple locations can be included by using **Add Locations** to the right of the **selectedMaterialTreeRootLocations** parameter. For more information on adding path locations using the location widget, see [Manipulating a Scene Graph Location Parameter](#).



NOTE: Two things that are not recorded when a look file is written: changes over time (only differences for the current frame are recorded) and deleted locations (locations cannot be removed by look files - for geometry, a similar effect can be achieved by setting its visibility to off).

Assigning a Look File to an Asset

The easiest way to assign a Katana look file to your asset is by using the Importomatic, see [Using the Importomatic](#). It is also possible to assign a Katana look file using LookFileAssign.

To assign a Look File using LookFileAssign:

1. Create a LookFileAssign node and connect it to the recipe.
2. Select the LookFileAssign node and press **Alt+E**.
The LookFileAssign node becomes editable within the **Parameters** tab.
3. Assign the scene graph locations of the 3D assets to the LookFileAssign **CEL** parameter (see [Assigning Locations to a CEL Parameter](#) under [Changing a Node's Parameters](#)).
4. In the **asset** parameter, enter the Katana look file to assign.



NOTE: When you bake a LookFile, CEL statement locations are automatically amended to be relative to the root of the LookFile bake. This means that although the full hierarchy of the target scene does not need to match the source scene, intermediate hierarchies must correspond.

For example, the assignment baked at -

/root/world/geo/someGeometry:

./geometry/subGeometry/subSubGeometry

- works applied to any locations with matching intermediate hierarchy. Assigning the LookFile to **//subSubScene** in the path below would work -

/root/newScene/newSubScene/geometry/subGeometry/subSubGeometry

- as would assigning the LookFile to **//newSubBranch** in the path below-

/root/newSubBranch/geometry/subGeometry/subSubGeometry

Assigning the LookFile to **//worldAssets** in the path below would not work, as the intermediate hierarchies do not match -

/root/worldAssets/someGeometry/geo

Resolving Look Files

A look file is assigned to a location in much the same way a material is assigned. An attribute on the location, **lookfile.asset** in this case, stores where to retrieve the look file without actually copying the details to that location. In order to apply the changes specified in the look file for a particular pass, use a LookFileResolve node. The alternate, and preferred method, is to use the LookFileManager node, see [Making Look Files Easier with the LookFileManager](#).

To resolve the look file for a particular pass:

1. Create a LookFileResolve node and connect it to the recipe at the point you want to resolve for a specific pass.
2. Select the LookFileResolve node and press **Alt+E**.

The LookFileResolve node becomes editable within the **Parameters** tab.

3. In the **passName** parameter, enter the look file pass to use.



NOTE: To force a reload for a look file that is being resolved, click **Flush Look File Cache** in the LookFileResolve's parameters.

Overriding Look File Material Attributes

When a Katana look file is assigned to a location, the details of where to find the look file are stored, not the contents of the look file itself. To retrieve the actual contents, a LookFileResolve or LookFileManager node is needed. These nodes enable you to select a pass, stored within the Katana look file, and retrieve the scene graph locations for that pass.

While this behavior has a number of advantages, scene specific overrides need access to the information within the look file. To make scene specific changes you bring in a look file's materials and then change those material locations. This is achieved with either the LookFileOverrideEnable or the LookFileManager nodes. For details on overriding with the LookFileManager node, see [Overriding Look Files](#).

To override a look file material using the LookFileOverrideEnable node:

1. Create a LookFileOverrideEnable node and connect it to the recipe.
The LookFileOverrideEnable node should be connected at some point downstream of a LookFileAssign node but before the look file is resolved.
2. Select the LookFileOverrideEnable node and press **Alt+E**.
The LookFileOverrideEnable node becomes editable within the **Parameters** tab.
3. Enter the name of the look file to override in the **lookfile** parameter.
4. Enter the look file's pass name to use in the **passName** parameter.
The materials within the look file are brought into the recipe and can be overridden.
5. Edit the material as needed. See [Editing a Material](#) for further details.

Activating Look File Lights and Constraints

Katana maintains a list of lights, cameras, and constraints at **/root/world** within the scene graph. When a Look File brings in a light or constraint, the lists at **/root/world** need to be updated. The LookFileLightAndConstraintActivator node activates Look File lights and constraints by updating the respective lists. It is also used to add constraints from LookFiles to the global constraint list. This list is used to specify the order in which constraints are evaluated, so this only has to be done if the constraints from the LookFile need to be evaluated in a specific order.

To activate lights and constraints from within a look file:

1. Create a LookFileLightAndConstraintActivator node and connect it to the recipe at some point downstream of a LookFileResolve or LookFileManager node.
2. Select the LookFileLightAndConstraintActivator node and press **Alt+E**.
The LookFileLightAndConstraintActivator node becomes editable within the **Parameters** tab.
3. Find the lights or constraints to activate by either:
 - selecting **Action > Search Entire Incoming Scene...**,
 - OR**
 - selecting a location within the scene graph and then selecting **Action > Search Incoming Scene From Scene graph Selection...**

Any look files with lights or constraints, found during the search, populate the node's hierarchical display (located below the **Action** menu in the **Parameter** tab).
4. Enable the lights and constraints for activation by right-clicking the **.klf** file in the hierarchical display and selecting **Enable** (or expanding the hierarchy and doing it individually).

Using Look Files as Default Settings

It is often desirable to have consistent default render settings across an entire show. Most render settings reside in the scene graph at **/root**. These settings can be stored in a Katana look file and brought in to each recipe of a show.

Creating a look file for a show's default settings is the same as creating any other look file but you need to have the look file record changes at **/root**, which is not recorded by default.

Saving Changes to /root as Part of a Look File

With the LookFileBake node's parameters in the **Parameter's** tab, open up the **options** parameter grouping and select **Yes** for the **includeGlobalAttributes** dropdown.

The look file now records changes to **/root**.

Setting a Globals Look File for a Recipe

Look files for assets are assigned to the location of the asset. As a look file for a show's settings is designed to repeat the changes made to **/root**, a LookFileGlobalsAssign node associates a look file with the **/root** location (this can also be achieved with the LookFileManager node, see [Assigning and Unassigning a Global Look File in the LookFileManager](#)).

To have a look file associated with **/root**:

1. Create a LookFileGlobalsAssign node and connect it to the recipe at the point you want to setup the show's default settings.

2. Select the LookFileGlobalsAssign node and press **Alt+E**.
The LookFileGlobalsAssign node becomes editable within the **Parameters** tab.
3. Enter the look file to use in the **asset** parameter.
4. If you want the look file to be resolved immediately, select **Yes** from the **resolveImmediately** dropdown.



TIP: You can force a reload of the look file at anytime by either: clicking the **Flush Look File Cache** button in the **Parameter** tab (when the LookFileGlobalsAssign node is editable), or by clicking  at the top of the Katana window.

Making Look Files Easier with the LookFileManager

The LookFileManager node has a lot of the functionality mentioned above, but it does it all in one node!

The LookFileManager node can:

- Assign a look file to **/root**, thus providing a show's default settings, in the same way as the LookFileGlobalsAssign node.
- Bring in a look file's material locations enabling them to be overridden, in the same way as the LookFileOverrideEnable node.
- Define which passes to resolve, in the same way as the LookFileResolve node. The LookFileManager node can resolve multiple passes, providing an output for each.

Connecting the LookFileManager Node into the Recipe

Create a LookFileManager node and connect it to the recipe at the point you want to resolve any look files into their respective passes.

Bringing a Look File into the Scene Graph

You can bring in a look file into the scene graph for later overriding or assigning to **/root** (to set a shot's global settings). This is done by adding the look file to the **Look Files** list of the LookFileManager node.

You can add a look file to the **Look Files** list in a number of ways:

Adding the Look File Currently Assigned to a Scene Graph Location

With one or more scene graph locations selected, right-click inside the **Look Files** list (or click ) and select **Add Look File Asset From Scene graph Selection**.

Adding a Look File from All the Look Files in the Current Scene Graph

1. Right-click inside the **Look Files** list (or click ) and select **Find All Look File Assets In Incoming Scene...** .
The **Find Look File Assets On Incoming Scene** dialog displays. The dialog is populated with all the look files in the current scene graph.
2. Right-click on a look file you want to add and select **Add Look File Asset**.
You can repeat this step for as many look files as you want to add.
3. Click **Close** when you have finished.

Adding a Look File from a List of All Look Files At or Below a Scene Graph Location:

1. With one or more scene graph locations selected, right-click inside the **Look Files** list (or click ) and select **Find All Look File Assets Beneath Selection In Incoming Scene...** .
The **Find Look File Assets On Incoming Scene** dialog displays. The dialog is populated with all the look files assigned at or below the scene graph location selected.
2. Right-click on a look file you want to add and select **Add Look File Asset**.
Repeat this step for as many look files as you want to add.
3. Click **Close** when you have finished.

Adding a Look File That is Not Assigned Anywhere Within the Scene Graph:

1. Right-click in the **Look Files** list (or click ) and select **Advanced > Add Look File Asset From Browser...** .
2. Select the look file within the browser and click **Accept**.
The look file is added to the LookFileManager **Look Files** list and assigned as the look file to **/root**. To unassign it, uncheck the **Add As Look File Root Asset** checkbox.

Assigning and Unassigning a Global Look File in the LookFileManager

You can replicate the behavior of the LookFileGlobalsAssign node inside the LookFileManager.

Assigning a Global Look File

Assigning a Look File to /root that is Not Currently in the Scene Graph

1. With the LookFileManager node's parameters in the **Parameters** tab, right-click in the **Look Files** list (or click ) and select **Advanced > Add Look File Asset From Browser...** .
The **Load Look File** dialog displays.
2. Select the look file within the browser and click **Accept**.
The look file is added to the LookFileManager **Look Files** list and assigned as the look file to **/root**.

Assigning a Look File that is Currently Within the Scene to /root

1. Bring the look file into the LookFileManager node's **Look Files** list.
2. Right-click on the look file (or select it and click ) and select **Use Look File For Scene Globals**.

Unassigning a Global Look File

It is possible to unassign a look file previously assigned to **/root** within the LookFileManager node without deleting it.

To unassign a look file from **/root**:

Within the **Look Files** list, right-click on the look file (or select it and click ) and select **Disable Use of Look File For Scene Root Attribute**.

Removing a Look File from the Look Files List

You can remove a look file from the **Look Files** list of the **LookFileManager** node. Removing a look file that has previously been assigned to the **/root** scene graph location unassigns it. Also, any look file that is removed from the **Look Files** list is no longer available for material overrides within the scene graph.

To remove a look file from the LookFileManager's **Look Files** list:

Within the **Look Files** list, right-click on the look file (or select it and click ) and select **Remove Look File From Manager**.

Managing Passes in the LookFileManager

Each look file has one or more passes. The LookFileManager can resolve as many of these passes as needed, creating an output for each (the **default** pass is always resolved). One technique is to have the look file that is assigned to **/root** contain all the necessary passes for that shot. This method means only one look file needs to be brought into the LookFileManager node to define all the passes that need resolving.

The **Passes** list to the right of the **Look Files** list inside the LookFileManager shows a list of passes that are both being resolved and are available within a look file to be resolved. Each pass name has one of three states:

-  - this pass is not only being resolved, the LookFileManager is the view node and the **Scene Graph** tab shows the results of resolving for this pass.
-  - this pass is being resolved, it has an output from the LookFileManager.
- no icon - this pass is within the currently selected look file but is not being resolved.

Having the LookFileManager Resolve Additional Passes

1. Within the **Look Files** list for the LookFileManager node, click on the look file with additional passes. The **Passes** list to the right of the **Look Files** list shows additional unresolved passes that are contained within the look file. These additional passes are displayed with no accompanying icon.
2. In the **Passes** list, right-click on the pass to resolve and select **Add Selected Pass Name Output**.
The pass is now resolved and an output is added to the LookFileManager node.

Changing Which Pass to Use When the LookFileManager is the Current View Node

- right-click on the pass in the **Passes** list and select **View Scene graph For Pass**, or
- select the pass in the **Passes** list and select  > **View Scene graph For Pass**, or
- click ▼ next to the pass name.

Overriding Look Files

When a look file is added to the **Look Files** list, its materials are added to the scene graph under the location **/root/materials/lookfile**. You can then override/edit these materials.

Overriding/Editing a Material Within a Look File

1. Add the look file to the **Look Files** list.
2. In the **Parameters** tab, select **Add Override > Material**.
You can narrow the list of nodes in the **Add Override** menu using the **Filter** field.
To have the new Material node override affect all passes, toggle the **New Overrides Active For All Passes** to on.
3. Follow the steps for overriding and editing a material at [Editing a Material](#).



NOTE: It is also possible to **Shift**+middle-click and drag a node into the overrides list from within the **Node Graph** tab.

Toggling the Ignore State of an Override

In the **Add Override** list, right-click on the override (or select it and click ) and select **Toggle Ignore State**.

Duplicating an Existing Override

In the **Add Override** list, right-click on the override (or select it and click ) and select **Duplicate Override**.

Viewing the Parameters for an Override in a Separate Panel

In the **Add Override** list, right-click on the override (or select it and click ) and select **Tearoff Parameters of Override...**

Deleting an Override

In the **Add Override** list, right-click on the override (or select it and click ) and select **Delete Override** (or with it selected, press **Delete**).



NOTE: You can change which passes the overrides are valid for using the **active for passes** menu to the right of **Add Override**.



TIP: Although the most common use of the **Add Override** menu is for adding material overrides, any kind of override may be created so long as the node has both an input and an output.

Scene Data

Scene Attributes and Hierarchy

In Katana the only data handed down the tree from one filter to the next is the scene graph data provided by iterators.

Some examples of attribute data include:

- 3D transforms such as translate, rotate, scale, or 4x4 homogeneous matrices.
- Camera data such as projection type, field-of-view, and screen projection window.
- Geometry data such as vertex lists.
- Parameter values to be passed to shaders.
- Nodes and connections for a material specified using a network.

Since all data is represented by attributes in the scene graph, any other data needed by the renderer (such as global options), or any data to be used by other Katana nodes downstream (such as material assignment), needs to be stored as attributes too. Examples include:

- Lists of available lights and cameras.
- Render global settings such as what camera to use, what renderer to use, the render image resolution, and the camera shutter open and close times.
- Per object settings such as visibility flags.
- Definitions of what renderer outputs (AOVs) are available.

Common Attributes

Attributes in the hierarchy can also make use of inheritance rules, so if an attribute isn't set directly at a location, it can inherit values set on a parent location.

It is worth spending some time using the **Scene Graph** and **Attributes** tabs in the UI to examine some of the attributes at different locations to get a feel for how data is represented and some of the common conventions.

/root holds settings needed by the renderer as a whole, such as flags to be passed to the command that launches the renderer or global options. For example:

- **renderSettings** : common settings for any renderer, such as:
 - **renderSettings.renderer**—what renderer to use.
 - **renderSettings.resolution**—the image resolution for rendering.
 - **renderSettings.shutterOpen** and **renderSettings.shutterClose**—shutter open and close times for motion blur (in frames relative to the current frame).
- Depending what renderer plug-ins you have installed you also see renderer specific settings, such as:
 - **prmanGlobalStatements**—Pixar's RenderMan specific global settings.
 - **arnoldGlobalStatements**—Arnold specific global statements.

Collections defined in the current project are also held as attributes at **/root**, in the **collections** attribute group.

By convention attributes set at **/root** are set to be non-inheriting.

/root/world is used as the base location of the object hierarchy where you can set default values to be inherited by all objects in the scene.

Common attributes include:

At any location in the world hierarchy:

- **xform**—the transformations (rotation, scale, translate or 4x4 matrices) to be applied at this level in the hierarchy.
- **materialAssign**—the path to the location of a material to be assigned at this location in the hierarchy.
- Renderer specific per-object options, such as tessellation settings and trace visibility flags. These are held in render specific attribute groups such as **prmanStatements** and **arnoldStatements**.

At geometry, camera and light locations:

- **geometry**:
 - for geometric objects this holds data such as vert-lists, UV co-ordinates, and topological data.
 - for cameras and lights this holds data such as the fov and projection type
 - **geometry.arbitrary** is used to hold arbitrary data to be sent to the renderer together with geometry, such as primvars in RenderMan or user data in Arnold.

At material nodes:

- **material**:
 - declarations of shaders and what parameters are set

The type of a location, such as whether it is a camera, a light or a polygon mesh, is simply held by an attribute called **type**. Common values for **type** include:

- **group** – for general group locations in the hierarchy.
- **camera** – for cameras.
- **light** – for lights.

- **polymesh** – for a polygon mesh.
- **subdmesh** – for a sub-division surface.
- **nurbspatch** – for a NURBS surface.
- **material** – for a location holding a material made of monolythic shaders.
- **network material** – for a location holding a material defined by network nodes.
- **renderer procedural** – for a location to run a renderer specific procedural, such as a RenderMan procedural.
- **scenegraph generator** – for a location to run a Katana Scene Graph Generator procedural.

One key idea is that it doesn't matter where the scene graph data comes from: all that matters is what the attribute values are. For example, geometry and transforms can come in from external files, such as Alembic, but it can also be created by internal nodes, or even AttributeScripts that use Python to set attribute values directly.

In principle you could create any scene with just a combination of LocationCreate and AttributeScript nodes, though you'd probably have to be a bit crazy to set your scenes up that way!

On a more advanced note, attributes are also used to store data about procedural operations that need to be run downstream, such as AttributeScripts deferred to run later in the **Node Graph**, or the set-ups for Scene Graph Generators and Attribute Modifiers. For instance, if you create an AttributeScript and ask for it to be run as a later process, such as during MaterialResolve, all the data about the script that needs to be run is held in an attribute group called **scenegraphLocationModifiers**.

The Process of Generating Scene Graph Data

As mentioned earlier, the real core of Katana is that what we want to render is described by a tree of filters, and that these filters are designed to be evaluated on demand. We're now going to look in a bit more detail at how Katana generates scene graph data.

The main interface that users have is the **Node Graph** tab. They create a network of nodes to specify things like Alembic files to bring in; create materials and cameras; set edits and overrides; and they can create multiple render outputs in a single project. The parameters for nodes can be animated, set using expressions, and manipulated in the **Curve Editor** and **Dope Sheet** views. The **Node Graph** can have multiple outputs and even separate disconnected parts, and it has potentially different parameter settings at any time on the timeline.

When we want to evaluate scene data, such as when doing a render or inspecting values in the UI, the nodes are used to create a description of all the filters that are needed. This filter tree has a single root, and represents the recipe of filters needed to create the scene graph data for the current frame at the particular node we are using for output.

It is this filter tree description that is handed to output processes such as RenderMan or Arnold. For the geekily inclined: this is actually done by handing a serialized description of the filter tree as a parameter to the output process, for example, a string parameter to a RenderMan or Arnold procedural.

The actual generation of scene graph data is done by using this description of the filters to create scene graph iterators. These are then used to walk the scene graph and access attribute values at any desired locations in the

Scene Graph tab. This approach of using iterators is the key to Katana's scalability and how all scene graph data can be generated on demand.

Using the filter tree, the first base iterator at **/root** is created. This can be interrogated to get:

- A list of the named attributes at that location.
- The value of any particular named attribute or group of attributes. For animated values there may be multiple time samples, with any sample relevant to the shutter interval being returned.
- New iterators for child and sibling locations to walk the hierarchy.

This process is also used inside the UI to inspect scene graph data when using the **Scene Graph**, **Attributes** and **Viewer** tabs. In the UI, the same filters and libraries that are used while rendering are called as the user expands the scene graph and inspects the results. This allows the user to inspect the scene graph data that is generated at any node for the current frame. TD's can use the UI as an IDE for setting up filters in a visual programming approach, and then running those filters to see how they affect the generated scene graph data.

The APIs are covered in more detail later, but the main API to create and modify the node graph is the Python NodegraphAPI, and the main ones to create new filters are the C++ Scene Graph Generator API and Attribute Modifier Plug-in API.

Structured Scene Graph Data

While Katana can handle quite arbitrarily structured scene graph data, there are a number of things worth considering both from the point of view of presenting good data to the renderer, as well as to enable users to work with the scene graph data in the user interface.

Bounding Boxes and Good Data for Renderers

When working with renderers that allow recursive deferred loading, the standard way that Katana works is to expand the scene graph until it reaches locations that have bounding boxes defined, then declare a new procedural call-back to be evaluated if the renderer asks for the data inside that box.

To make use of deferred loading these bounding boxes should be declared with assets, and nested bounding boxes should be structured so that only what is needed has to be evaluated. For instance, if you have a cityscape where only the top of most buildings is seen by the renderer, it is inefficient to have just a single bounding box for the whole of each building. This is because a lot more geometry than needed is declared to the renderer whenever the top of a building is seen.

There is an optional attribute called **forceExpand** that can be placed at any location to force expansion of the hierarchy under that location rather than stopping when a bounding box is reached. This can be useful when you know that the whole of the contents of a bounding box are going to be needed if any part of it is requested. There are also times when it is more efficient to simply declare the whole scene graph to a renderer than use deferred evaluation, such as if you are calculating the global illumination for a scene that you know can fit into memory. In

particular, some renderers can better optimize their spatial acceleration structures if they have all of the geometry data in advance rather than using deferred loading.

Proxies and Good Data for Users

Since users are working with scene graph data in Katana it's also good to consider things that may help them navigate and make sense of the scene.

The bounding boxes used by the renderer can also help provide a simplified representation in the Viewer of the contents of a branch of the hierarchy when the user opens the scene graph to a given location.

To give an even better visualization you can register proxies at any location which are displayed in the **Viewer** but not sent to a renderer.

Ops can be used to define viewer proxies on scene graph locations. Two main attribute conventions are currently supported:

- **ViewerProxyLoader (legacy mode)** - An Alembic cache can be loaded through the default ViewerProxyLoader, setting the **proxies.viewer** string attribute on the target location to the path to the relevant **.abc** file. You can also customize Katana to read proxies from custom data formats by creating a Scene Graph Generator to read the relevant file format and using a plug-in for the **Viewer** that simply declares which Scene Graph Generator to use for a given file extension.
- **Op-based** - Ops can be chained to create the geometry to be used as a proxy by adding group child attributes to the **proxies.viewer** group attribute on the target location. Each child group attribute represents an Op and its content must contain:
 - a string attribute named **opType** defining the type of the Op to be used.
 - a group attribute named **opArgs** containing attributes defining the Op arguments.

Here's an example of the attributes hierarchy using two Ops to generate the proxy geometry:

```
Location
  /root/world/geo/group

Attributes:
  ...
  proxies
    viewer
      proxyOp_1
        opType 'AlembicIn' (StringAttribute)
        opArgs
          fileName '/tmp/myProxy.abc' (StringAttribute)
      proxyOp_2
        opType 'Messer' (StringAttribute)
        opArgs
          displacement 0.23 (DoubleAttribute)
```

...

Proxy caches are considered animated by default. If the proxy file has animation, that is used by default, but you can also explicitly control what frame from a proxy is read using these additional attributes: **proxies.currentFrame**, **proxies.firstFrame**, and **proxies.lastFrame**. Static proxy caches can be defined by setting the **proxies.static** IntAttribute to **1**.

To help users navigate the scene graph, group locations can be indicated as being **assemblies** or **components**. These terms originate from Sony Pictures Imageworks where they are used to indicate whether an asset is a building block component or an assembly of other assets. In Katana's user interface they are simply used as indicators for locations that are good for the user to open the scene graph up to. In the **Scene Graph** tab there are options to open to the next assembly, component, or level-of-detail level, and double-clicking on a location automatically opens the scene graph to the next of these levels.

For the user it's useful if proxies or bounding boxes are at groups indicated as being **assemblies** or **components**, so the user can open the scene graph to those levels and see a sensible representation of the assets in the **Viewer**.

To turn a group location into an **assembly** or **component** the **type** attribute at that location simply needs to be set to **assembly** or **component**. If you are using ScenegraphXML, there is support for indicating locations as being **assemblies** or **components** within the ScenegraphXML file.

In general it also helps users if the hierarchy isn't too 'flat', with groups containing a very large number of children. Structure can help users navigate the scene graph.

Level-of-Detail Groups

Levels of Detail (LODs) is used to allow an asset to have multiple representations. Katana can then be used to select which representation is used in a given render output.

Conventionally LODs are used to hold a number of asset versions each with a different amount of geometric complexity, such as a high level of detail to use if the asset is close to the camera and middle and low levels of detail if the asset is further away. By selecting an appropriate version of each asset to send to the renderer the overall complexity of a shot can be controlled and render times managed.

In Katana, LODs can also be used to declare completely different versions of an asset for different target outputs, such as a bounding volume representation for a volumetric renderer in addition to standard geometrical representations, such as polygon meshes, to be used by conventional scanline renderers or ray-tracers.

Multiple levels of detail for an asset are declared by having a **level-of-detail group** location which has a number of **level-of-detail** child locations. Each of these child locations has metadata to determine when that level of detail is to be used. Under each of these locations you have a separate branch of the hierarchy that declares the actual geometry used for that LOD representation of the asset.

The most common metadata used to determine which level of detail to use are tags or weights. Tags allow each level of detail to be given a 'tag' name with a string. Selection of which level of detail to use can be done using this tag name, such as select the level of detail called 'high' or 'boundingVolume'.

Weights allow a floating point value to be assigned to each level of detail. Selection can then be done by choosing the closest level of detail to a given weight value. This allows sparse population of levels of detail, for example not every asset might have a 'medium' level of detail, but if you select them by weight then the most appropriate LOD from whatever representations exist can be selected.

The LodSelect node can be used to select which one of the LODs to use using either tags or weight values. This uses a **CEL** expression to specify the LOD groups you want to base the selection on.

Some renderers, such as Pixar's RenderMan, have features to handle multiple LODs themselves. Selection of which LOD to use, and potential blending between the LODs as they transition, is done at render-time. This is specified by having range data associated with each LOD that describes the range of distances from camera to use that LOD for, and the transition range for any blending. LOD range data can be set up using the LodGroupCreate or LodValuesAssign nodes

Alembic and Other Input Data Formats

It is possible to bring in 3D scene data from any source. However, due to the way that filters can get called recursively on-demand, it is best to work with formats that can be efficiently accessed in this manner. This is one of the reasons that we recommend Alembic as being ideally suited for delivering assets to Katana.

If you want to write a custom plug-in to read in data from a new source, such as using an in-house geometry caching format, you can write a Scene Graph Generator plug-in. This is a C++ API that allows you to create new locations in the scene graph hierarchy and set attribute values at those locations.

ScenegraphXML

ScenegraphXML is provided with Katana as a simple reference format that can be used to bring structured hierarchical scene data into Katana using a combination of XML and Alembic files. One example of how it can be used includes using Alembic to declare some base 'building block' components and then use XML files as 'casting sheets' of which assets to bring in.

ScenegraphXML files can also reference other ScenegraphXML files to create higher level structured assets. For instance you could have a multiple level hierarchy where buildings are built out of various standard components, and sets of buildings are assembled together in to city blocks, and city blocks are assembled together into whole cities.

In the hierarchy created by ScenegraphXML locations can be set to be **assemblies** or **components** to help users navigate the scene graph. By default any reference to another ScenegraphXML creates a location of type **assembly** and any reference to an Alembic file creates a location of type **component**. You can also override this behavior by directly stating the type of any location in the ScenegraphXML file.

You can also register proxies at locations, and create LOD groups.

Graph State Variables

Graph State Variables can be used to control which nodes in the node graph contribute to scene graph processing, based on the values of user-set variables. These values can be set either at the whole project level or by nodes in the node graph.

Graph State Variables have been designed to make it easier to set up a single Katana project, for instance to control the lighting for a whole sequence, or to perform edits and overrides only active in the node graph, depending on which output is currently being rendered. You can define your own variables and they can be used in many different ways.

The key concept is that Graph State Variables can be set either at a global, whole-project level, such as a variable for the shot number in a sequence that is being worked on, or at a local level using VariableSet nodes, such as for a variable that says which render pass is being evaluated.

You can then use VariableSwitch or VariableEnabledGroup nodes to control which nodes are active or not, based on the values of Graph State Variables. For instance, you could have a VariableSwitch node with different inputs based on the shot number, so which input is read depends on which shot in a sequence you are working on. Another example is that you could have an override for some attributes in a VariableEnabledGroup node, based on which render pass is being evaluated, so the override is only applied for specific output passes.

The essential idea is that Graph State Variables allow you to define the context in which the scene is currently being evaluated, and have nodes whose behavior can be changed depending on that context. You can also read the values of Graph State Variables in OpScripts or your own Op plug-ins in order to modify their behavior, based on the values of the Graph State Variables.

Setting Graph State Variables

Global Variables

Variables set using the **Project Settings** tab affect all node graph branches in the entire project and are, therefore, referred to as global.

To define a new global Graph State Variable:

1. Open the **Project Settings** tab and locate the **variables** group parameter.
2. Click the **Graph State Variables** menu on the right of the group header and choose **Add Variable**.
3. Click the wrench  menu to the right of the newly-created variable and choose **Rename**. Enter a variable name and click **OK**.

4. Enter a value for the variable in the dropdown widget. You can add new values while retaining old ones as options.

Katana also shows a **variables** widget in the main menu bar. This cannot be used to define a new variable, but can be used to change the value of an existing global variable. To do this, locate the variables widget in the main menu bar, for example , and left-click.

Local Variables

Variables set using a VariableSet node are referred to as local and affect the Graph State seen by nodes upstream of the VariableSet node. For a VariableSet node to have an effect, it must be a contributing node, that is to say, upstream of the viewed node and not disconnected (by a Switch node, for example).

The variable name used by a VariableSet node need not already exist as a global Graph State Variable. If a global variable of the same name does exist, upstream nodes see the new value and consider it local. This implies that changing the global variable's value through the **Project Settings** tab or the **variables** menu bar entry has no effect on the value seen by nodes upstream of the VariableSet.

A Graph State Variable can be deleted by a VariableDelete node, which again only affects the upstream Graph State.

Inspecting Graph State Variables

The value of a project-wide Graph State Variable can be inspected through the **variables** section of the main menu bar. The value of a local Graph State Variable can be inspected in the **Parameters** tab. To do this:

1. Set the edit flag on the node at which you'd like to view the Graph State Variables.
2. Set the view flag on whichever downstream node you'd like to produce scene data from.



NOTE: If you have a VariableSet node that does not lie between the edited and viewed nodes, it is not a contributing node and has no effect.

3. In the **Parameters** tab, click the **Graph State Variables** button () to the right of the node name.

A list of variables and their current values appears in the pop-up menu. The values of Graph State Variables cannot be changed in the widget; it is read-only.

When the **Graph State Variables** menu is shown while a node is currently viewed, and the edited node is part of the active node graph, the menu displays Graph State Variables that are seen in the portion of the node graph between the currently-viewed node and the respective edited node.

When the **Graph State Variables** menu is shown while no node is currently viewed, or if the edited node is not part of the active node graph, the menu displays a label with red text to indicate that the edited node is not part of the currently-active node graph and that Graph State Variables are not available.

To see the effect of VariableSwitch and VariableEnabledGroup nodes, open the **Node Graph** tab and choose **Edit > Dim Nodes Not Contributing to Viewed Node** from the tab's menu, or press **Alt+.** (full stop). When this option is turned on, nodes in the **Node Graph** tab that are not part of the currently-active node graph portion are drawn in a dimmed appearance. This is determined by taking into account the active graph state and possible VariableSwitch and Switch nodes that may be a part of the node graph.

Reading Graph State Variables

Nodes

The following Katana node types perform some logic, based on the Graph State Variables passed to them.

VariableSwitch

This node type is similar to the Switch node type, which uses a parameter to determine which input port should be followed. VariableSwitch nodes make this determination by reading a Graph State Variable and attempting to match its value against patterns defined for input ports, or input port names directly, should a port have no pattern define. For example, a VariableSwitch node could be configured to read a "levelofdetail" variable, with input ports named "high" and "low". The same effect can be achieved by defining the following patterns:

- i0 → "high"
- i1 → "low"

A pattern takes the form of a CEL statement, where the {@[name]=="value"} syntax may be used to specify requirements of additional Graph State Variables.

If no port matches the value of the Graph State Variable, the default behavior is to use the left-most input. If more than one pattern matches, the default behavior is to use the left-most matching input.



NOTE: If a VariableSwitch node defines no patterns, input selection is performed using a faster look-up operation. This may be useful for nodes with a large number of input ports.

VariableEnabledGroup

The VariableEnabledGroup node type is an extension to the Group node type, which allows you to combine multiple nodes into a single unit. VariableEnabledGroup bypasses its internal nodes entirely, unless a given Graph State Variable matches a pattern. For example, if the group contained nodes responsible for material assignments, the node could be configured to read an "assignmaterials" variable, with the pattern set to "yes". The material assignments would then only be active if the value of "assignmaterials" was set to "yes".

Scripts

OpScript

Graph State Variables can be read in OpScript nodes. The function signature for this is:

```
string Interface.GetGraphStateVariable(string variableName)
```

OpScripts cannot manipulate Graph State Variables as, by the time an OpScript is executed, the contributing nodes (and their associated Graph States) have already been determined.

The `getGraphState()` Function

Instances of **NodegraphAPI.Node** have a **.getGraphState()** method for retrieving the local graph state seen by the node. This method takes two optional arguments:

- **node** - a **NodegraphAPI.Node** instance. This is used as a starting node for walking the node graph. If **None** or not given, this parameter defaults to the currently-viewed node.
- **graphState** - a **NodegraphAPI.GraphState** instance. This is the global graph state that is passed to the start node (given above). If **None**, or not given, this parameter defaults to the scene-wide global graph state.

Additionally, the global graph state can be retrieved through **NodegraphAPI.GetCurrentGraphState()**.

How Do Graph State Variables Work?

When evaluating scene graph data at any node, Katana follows a recursive process of asking the node to describe its inputs, then following those inputs and repeating the procedure on any nodes above. This process has the effect of identifying the nodes that contribute to the scene, which can be evaluated later to produce the scene graph. Many nodes always identify the same inputs, but a few use conditional logic. For example, a Switch node uses a parameter to choose its input; sub-graphs above non-active inputs are never evaluated.

Katana also maintains a Graph State data structure when traversing up the node graph. This contains information such as the current frame and the shutter timings. As part of identifying their inputs, nodes can read from and write to the Graph State. For example, a TimeOffset node reads the current frame time and increments or decrements it by some amount. The modified Graph State is then passed to the node above. It is important to realize that the Graph State information flows up the node graph, rather than down, as scene data does.

Graph State Variables essentially allow us to define key-value pairs within the Graph State, and can be set at the project or node level. They can then be referenced and manipulated by other nodes, allowing for a powerful workflow feature, where groups of nodes and entire node graph branches can be enabled and disabled with ease.

Resolvers

Overview

Resolvers are filters that must be run before actual rendering, in order to get data into the correct final form. Typically they are used for things like material assignments, executing overrides, and calculating the effects of constraints.

The only data that can be passed from one filter to the next is the scene graph, with its attributes. Resolvers are procedural processes that can be executed with just attribute data, which allows us to separate executing procedural process into two stages:

1. Set up appropriate attributes in the scene graph that define what process to run and any parameters that need to be handed to the procedural.
2. Run a resolver that reads those attributes and executes the procedural process.

This separation into two stages gives a lot more flexibility than if all procedural processes had to be executed immediately. Because they are only dependent on the correct attributes being set at locations in the scene the configuration to set up the process can be done in a variety of different ways.

For instance, material assignment is based on a single string attribute named **materialAssign**, which gives the path to the location of the material to be used. This attribute is then used in a resolver called **MaterialResolve**, which takes the material from the path in the **materialAssign** attribute and creates a local copy, with all the relevant attributes set to their correct values (taking into account things like material overrides). Because **MaterialResolve** only looks for an attribute called **materialAssign**, material assignment can be set up in a number of different ways:

- Using **MaterialAssign** nodes, which set the **materialAssign** attribute at locations that match the CEL expression on the node.
- Using an **AttributeScript** to set the value of **materialAssign** using a Python script.
- Using a custom C++ procedural, such as a **Scene Graph Generator** or **Attribute Modifier**, to set the values of **materialAssign**.

You can also use a **LookFile** that resolves the correct value for **materialAssign** onto given objects.



NOTE: Using a LookFile, material assignment is resolved, rather than just setting the **materialAssign** attribute. The **materialAssign** attribute does not survive the processing that occurs in a LookFileResolve node.

Resolvers allow us to keep the data high-level and user meaningful as possible since until the resolver runs the user can directly manipulate the attributes that describe how the process should run instead of only being able to manipulate the data that comes out of the process.

For instance, since material assignment is based on the **materialAssign** attribute we can:

- Change what material an object gets by changing that one attribute's value.
- Change the material on every object that is assigned a specific material by changing the attributes of the original material.

In essence resolvers manipulate the parameters of the process, rather than just the data that comes out of the process, with access to all the tools available in Katana for inspecting, modifying and overriding attributes.

Examples of Resolvers

As well as MaterialResolve there are a number of other common resolvers:

- **ConstraintResolve.** This evaluates the effect of a constraint on the transform of a location.
- **LookFileResolve.** This replays the changes described in a look file back onto an asset. This is probably the resolver that users are most likely to be directly exposed to if they don't use the LookFileManager as they are directly using LookFileResolve nodes.
- **ScenegraphGeneratorResolve.** This executes Scene Graph Generators, custom procedurals which create new scene graph data. This resolver runs on any location of type **scenegraph generator**, and looks for an attribute named **scenegraphGenerator.generatorType** that specifies the **.so** to use.
- **AttributeModifierResolve.** This is similar to ScenegraphGeneratorResolve but executes Attribute Modifier plug-ins, custom procedurals that can modify attribute values. It looks for any location with an attribute called **attributeModifiers.xxx**.



NOTE: AttributeScripts are actually a type of Attribute Modifier, so AttributeModifierResolve can also be used to execute deferred Attribute Scripts.

Implicit Resolvers

Resolvers can be run by putting nodes explicitly into a project, but there are also a standard set of resolvers that are automatically implicitly run before rendering. In effect these are nodes that are automatically appended to the root

of a node graph before rendering, so that it's not necessary to manually add all the needed resolvers. This allows execution of procedural processes that are always needed, such as `MaterialResolve`.

The standard implicit resolvers are:

- `AttributeModifierResolve` (`resolveWithIds=preprocess`)

This resolves and Attribute Modifier plug-in (including `AttributeScripts`) that have their `resolveld` set to 'preprocess'

- `MaterialResolve`

As previously described, this looks for **materialAssign** attributes and creates local copies of materials taking into account any material overrides. It also executes any Attribute Scripts scripts set to execute **during material resolve**, allowing scripts to be placed on materials that affect their behavior every time they are assigned to a different location.

- `RiProceduralResolve`

This is similar to `MaterialResolve` but for renderer procedurals, such as `RenderMan` or `Arnold` procedurals. It looks for any locations with **rendererProceduralAssign** attributes,

- `ConstraintResolve`

This looks for any constraints defined at `/root/world` in **globals.constraintList** and calculates the effects of any constraint on the transforms of locations.

- `AttributeModifierResolve` (`resolveWithIds=all`)

This resolves any Attribute Modifier plug-in that hasn't been resolved previously.

Normally when you inspect scene data in Katana's UI you see the results before the implicit resolvers are run. It's only at render time that the implicit resolvers are added. If you want to see the effect of the implicit resolvers on the scene data you can switch them on by clicking on the **Toggle Scenegraph Implicit Resolvers** clapper-board icon in the menu bar or at top right hand side of the **Scene Graph**, **Attributes** or **Viewer** tabs. It then glows orange and a warning message is displayed to indicate that the implicit resolvers are now active in the UI.

For instance, if you switch the implicit resolvers on and view the attributes at a location that has an assigned material you'll see that:

- There is now an attribute group called **material** with a local copy of the assigned material.
- Any material overrides have been applied to the shader parameter values.
- The original **materialAssign** value is removed.
- Similarly any **materialOverride** attributes are removed.
- The values of **materialAssign** and **materialOverride** are copied into **info** so they can still be inspected, for reference, but they are no longer active.

Creating your own resolvers

You can use `AttributeScripts` or custom Attribute Modifier plug-ins to create your own resolvers, including having them run implicitly .

There are a number of modes available for AttributeScripts and AttributeModifiers execution. These are controlled by the **resolveids** values in the attributes. For AttributeScripts there are four modes available from the UI:

- **immediate**: the script is resolved immediately after being set in the node.
- **during attribute modifier resolve**: the script is resolved by the first AttributeModifierResolve node it encounters, either directly in the node graph or by the AttributeModifierResolve implicit resolver if the users doesn't put any in.
- **during katana look file resolve**: the script is resolved during LookFileResolve when the changes from look files are written back on to assets.



NOTE: The LookFileManager includes LookFileResolve nodes.

- **during material resolve**: the script is executed as local copies of materials that are created on any location with assigned materials. This mode is designed for placing scripts on materials to customize how that material behaves when it is applied to an object (such as to randomize a material's shader settings every time it's applied).

Handling Textures

Introduction

Because textures are handled in a variety of different ways by shader libraries and studio pipelines Katana doesn't enforce rigid standards for how textures are declared, but acts as a flexible framework with some common conventions.

In particular there is a convention to use string attributes with the naming convention `textures.xxx` where `xxx` is the name of the file path for the texture. For example `textures.ColMap` would specify the filepath for a texture called `ColMap`.

Texture Handling Options

Materials with Explicit Textures

The simplest way of specifying textures is to have separate materials which each explicitly declare the textures they need to use as string parameters of the shaders. Each object that needs a different texture is simply assigned the relevant material.

Though this is simple it lacks flexibility. In particular it's common to want to be able to use the same material on multiple objects, but with each object picking up its own textures.

Using Material Overrides to Specify Textures

If you have exposed parameters on a material that define the textures to use, you can use material overrides to create new object specific versions of materials with the relevant textures.

The material that is to be used in common on a number of objects can be assigned to those objects (or assigned higher up the hierarchy and inherited), and a material override set on the objects to override shader parameters that specify textures with new object specific values.

This can be done directly using MaterialAssign nodes, but since all MaterialAssign nodes do is create attributes under a group called materialOverride we can also set up material overrides by directly

setting those attributes directly by any other process, such as using AttributeScripts.

For instance, this fragment of AttributeScript reads the attribute value contained in **textures.SpecMap** and use it to override an Arnold shader with a parameter called **SpecMapTexture**:

```
SpecMapPath = GetAttr('textures.SpecMap') SetAttr
('materialOverride.arnoldSurfaceParams.SpecMapTexture', SpecMapPath)
```

This means that a new copy of the material is created for the object with the shader's **SpecMapTexture** parameter changed to the appropriate values.



NOTE: Material overriding actually takes place as part of MaterialResolve, one of Katana's implicit resolvers. During MaterialResolve it looks for attributes in the 'materialOverride' group, and creates a new copy of the material at that location with the relevant changed to shader parameters.

Using the {attr:xxx} Syntax for Shader Parameters

There is also a special way of declaring string shader parameters to use the value of another attribute. If you define any string parameter on a shader to be **{attr:xxx}** then during MaterialResolve it looks for an attribute called **xxx** at the location the material is being assigned to and used that as the shader value.

For instance, if you have an Arnold image reader shader with a parameter called **filename** and you set **filename** to **{attr:textures.SpecMap}**, **filename** is set to the value of the attribute **textures.SpecMap** on any location the material is assigned to. This means you can set up the original shader to automatically pick up relevant texture name attributes for every object it is applied to. For example, create an Arnold Network Material with an ImageRead node, and have the filename parameter of the ImageRead node linked to a **textures.ColMap** attribute on geometry locations:

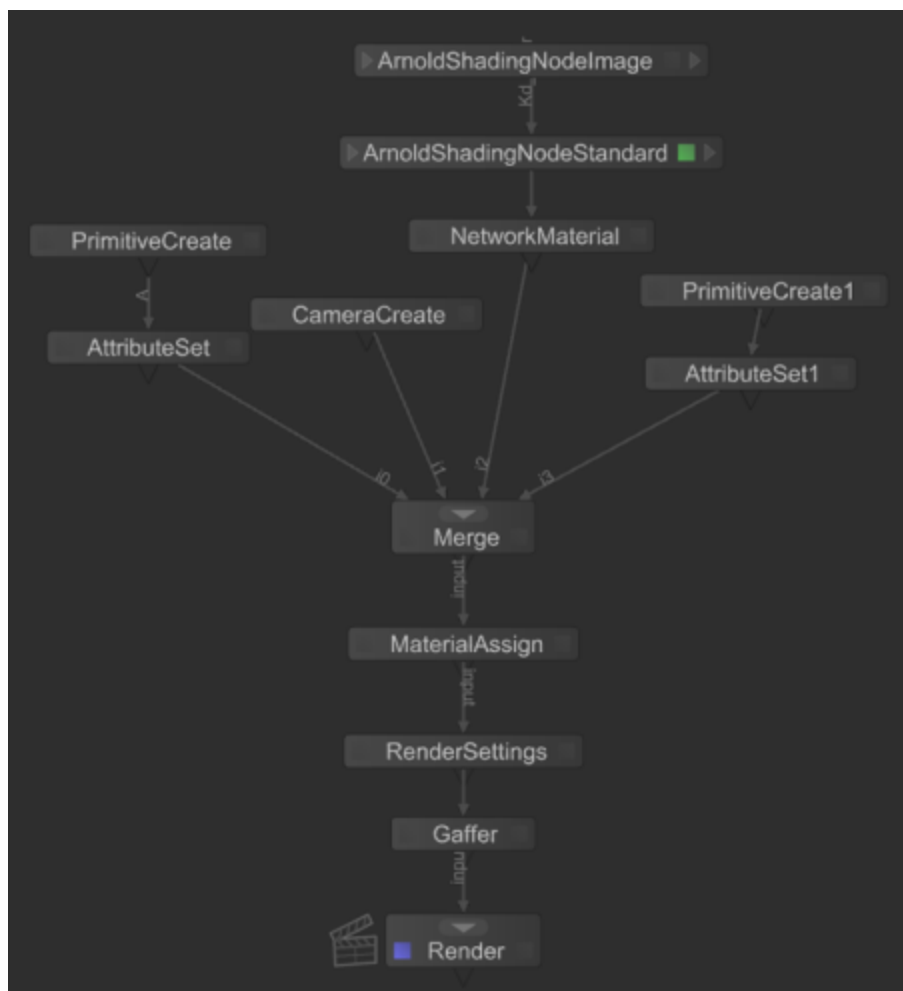
1. In an empty Katana scene create two ArnoldShadingNodes, set one to type **image**, and one to type **standard**. Link the output of the **image** node, to the **Kd_Color** parameter of the **standard** node.
2. In the **filename** field in the Image node's parameters enter:
`{attr:textures.ColMap}`
3. Add a NetworkMaterial node, with an Arnold standard terminal. Connect the output of the ArnoldShadingNode of type **standard** to the input of the NetworkMaterial node.
4. Add two PrimitiveCreate nodes, and two AttributeSet nodes. With the AttributeSet nodes set a string attribute with the attributeName textures.ColMap on each primitive location. Set the stringValue for each to the path to a texture (for example /tmp/yourTexture.tx and /tmp/yourOtherTexture.tx).
5. Add a CameraCreate node.
6. Connect the outputs of PrimitiveCreate nodes, the CameraCreate, and the NetworkShading node to inputs on a Merge node. Add a MaterialAssign node below the Merge node. Assign the NetworkShading material to each applicable scene graph location. For example, use the CEL statement:

```
/root/world/geo//*
```

Assignments created using **{attr:yourParameter}** are evaluated during material resolve. Therefore any material using those parameters must be explicitly assigned to any relevant scene graph locations, rather than relying on inheritance.

7. Add a RenderSettings node below the MaterialAssign node, and set the renderer to **arnold**.
8. Add a GafferThree below the RenderSettings node, set it to the **arnold-c** profile, and add a spotlight.
9. Add a Render node below the GafferThree node.
10. Position the spotlight, and the camera.
11. Right-click on the Render node and select **Preview Render**.

At material resolve time Katana picks up the **texture.ColMap** parameter on each of the geometry locations, and creates an instance of the assigned material for each, with the material's filename parameter set to the value of **textures.ColMap**.



NOTE: Because **{attr:xxx}** is evaluated during MaterialResolve you must apply the base material directly to every object that needs it, rather than using material inheritance in the hierarchy.

Using Primvars in RenderMan

RenderMan has a feature called primvars that allow you to directly override the value of any shader parameter.

This means that you can have a single instance of a shader used by multiple pieces of geometry (for example by being placed high up in the hierarchy) with string parameters for textures, but each piece of geometry can use its own specific textures by simply creating a primvar with the same name as shader parameter.

In Katana any string attribute called **textures.xxx** is automatically written out to RenderMan as a primvar called **xxx**. For instance if your shader has a string parameter called **BumpMap**, setting an attribute called **textures.BumpMap** on a piece of geometry creates a primvar called **BumpMap** on the geometry with the value of the **textures.BumpMap** attribute.

This means that with suitably named shader parameters you can simply create attributes in Katana called **textures.xxx** on pieces of geometry to allow each piece of geometry to pick up its individual textures.

You can also do per-face assignment of textures using primvars. If **textures.xxx** is set to an array of string values, with the number of elements matching the number of faces, that array is written out as a **varying** primvar instead of a **constant** one so each face picks up its own value.



NOTE: The **xxx** values must take the form of an array. For example, rather than:

```
SetAttr( 'textures.ColMap', "newTexture" )
```

use:

```
SetAttr( 'textures.ColMap', [ "newTexture" ]
```

Using User Custom Data

Some other renderers don't have RenderMan style primvars, but allow some form of custom user data that can be looked up by shaders. With a little more work and suitable shaders these can be used to give similar results.

For instance, in Arnold if you have shaders designed to look for user data that contain strings declaring the paths to textures, instead of the paths to the textures being direct parameters on the shaders, you can use user data to have a shared material on multiple objects and each object picks up its own individual textures.

Any string attribute called **textures.xxx** is automatically written out to Arnold as a piece of string user data called **xxx**, which can then be looked up inside shaders.

You can also do per-face assignment of textures using user data. If **textures.xxx** is set to an array of string values, with the number of elements matching the number of faces, that array is written out as a per-face array of user data so each face can pick up its own value.

Using Pipeline Data to Set Textures

Different pipelines often use different methods to specify which textures should be used on a particular asset. The normal convention in Katana is to use attributes called `textures.xxx` on geometry to hold the individual texture paths needed for that piece of geometry. That data can be set in a number of different ways. For instance:

Metadata on Imported Geometry

Arbitrary metadata can be read in with geometry on import, such as string data containing the texture paths that is written out into Alembic and read in as arbitrary geometry data. This means that assets can be created with additional metadata, such as by adding string attributes to shape nodes in Maya before writing the data to Alembic.

In Katana the convention is for arbitrary geometry data to be read in as attributes called **`geometry.arbitrary.xxx`**, which are then by default also written out as user or primvar data to renderers. This means that if you are using primvars or user data to specify textures you can have this work automatically.

Metadata Read in from Another Source

If texture assignment is specified by other sources in the pipeline, such as by having separate XML files associated with assets that give the texture paths to be used on any named piece of geometry, that metadata can be added to objects using `AttributeModifiers`.

Katana come with an example Attribute Modifier called `AttributeFile` that reads data from a simple XML format to create new attributes at locations in the scene graph. One of the demo scenes, `houseScene_textured.katana`, makes use of this Attribute Modifier together with an additional `AttributeScript` to take the attribute values read in and do some additional processing to turn them in to the final texture file paths.

Processes to Procedurally Resolve Textures

You could also use a resolver to procedurally set the values of **`textures.xxx`** to appropriate file paths, allowing the actual creation of these file paths as one of the last automatic processes in the pipeline.

The **`robot_textured.katana`** example demo project illustrates how metadata that comes in with a `ScenegraphXML` asset can be further processed by an `AttributeScript` to turn it into final texture paths. By setting these in attributes called **`textures.ColMap`**, **`textures.SpecMap`** and **`textures.BumpMap`** these are exported to RenderMan as primvars.

Asset Management

An asset is an item of data that contributes to a Katana project, such as an Alembic file, or material shader. A Katana project itself can also be an asset. An asset may have multiple versions, for example, incremental versions recording the history of a Katana recipe, and there may even be different meta-versions (or tags) of an asset indicating different purposes (such as lighting or animation).

Katana communicates with asset management systems through an asset plug-in. Assets are published to and retrieved from an asset management system, which handles their cataloging, storage. Crucially, as each saved version of an asset is stored with its version data, you can return to any saved point in an asset's history.

Asset Plug-ins

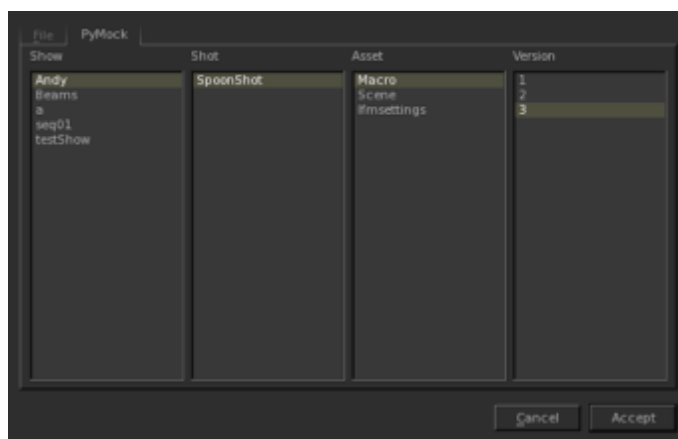
Katana includes an Asset API for plug-in authors, which consists of the following four core mechanisms:

- **Script Level Hooks**

Script level hooks for performing the **Pre-Publish** and **Post-Publish** asset management steps from within Katana. See [The Asset Publishing Process](#) for more on this.

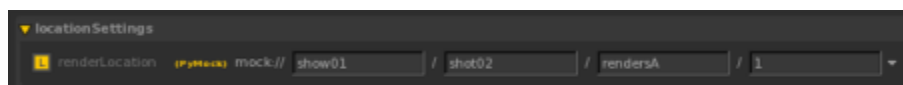
- **Browser Customization**

A mechanism for studios to replace the standard Katana file browser with a custom asset browser. For example, the browser used in the PyMockAsset example has fields for **Show**, **Shot**, **Asset**, and **Version**. For more on the PyMockAsset example plug-in, see [Example Asset Plug-in](#).



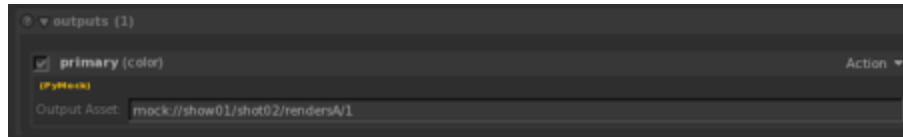
- **Parameter Display**

A mechanism for controlling the representation of an asset in Katana's **Parameters** tab.



• Render Output

A mechanism for controlling the representation of a render output's Asset ID in a Render node's **Parameters** tab.



The Asset Publishing Process

Publishing an asset from Katana performs the following steps:

1. Pre-Publish

Takes identifying information from you (for example which show, shot, asset and version) and passes that information to the asset plug-in. The plug-in returns an Asset ID - in the form of a string - to use in the **Publish** step.

2. Publish

Katana passes the Asset ID returned at the **Pre-Publish** stage to the asset management system, which resolves that ID to a file path. Katana generates the asset, and saves it to the resolved path.

3. Post-Publish

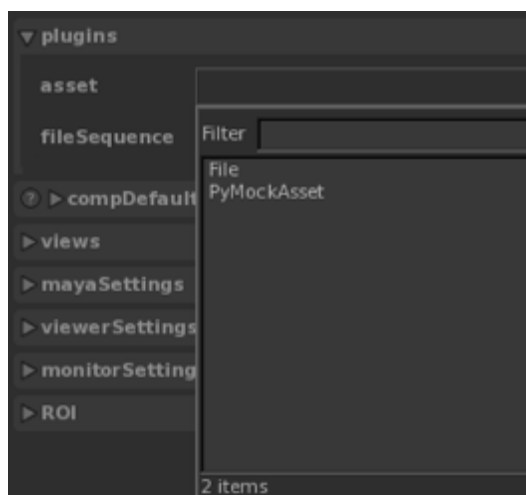
The asset management system handles storing the asset, and returns the Asset ID actually applied (which can be different to the one supplied in the **Pre-Publish** step). Katana uses that Asset ID from then on to identify the current version of the asset.

Choosing an Asset Plug-in

You can have multiple asset management plug-ins installed, but only one active at a time. Selecting which asset management plug-in to use is done in the **Project Settings** tab. Choose **Group > plugins > asset** and choose a plug-in from the dropdown list.



NOTE: The option **Project Settings** tab > **Group > plugins > asset > File** selects Katana's default, manual file management, rather than any asset-managed file management.



Example Asset Plug-in

Katana ships with an example Asset plug-in, PyMockAsset. To use the example plug-in the following path must be available in your KATANA_RESOURCES environment variable:

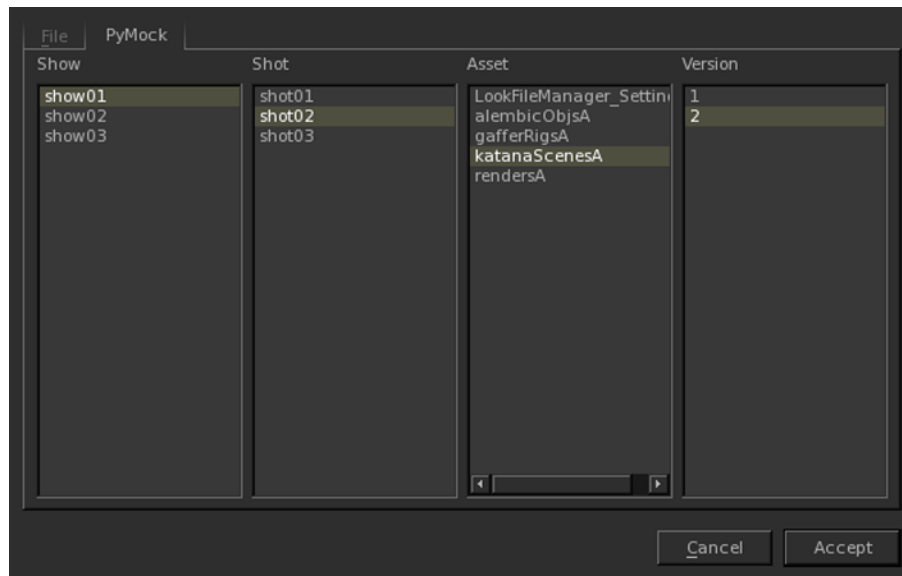
```
${KATANA_ROOT}/plugins/Resources/Examples/
```

PyMockAsset takes information on show, shot, asset, and version, and uses a browser customized with fields to hold that data. All of the images used in this chapter show the PyMockAsset plug-in.

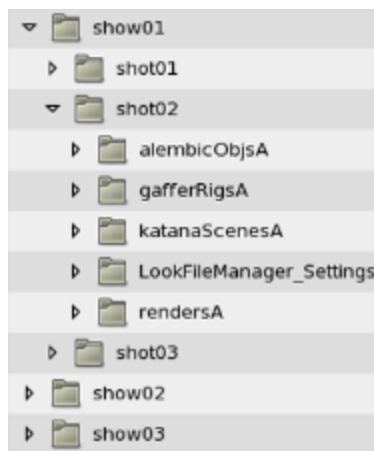
The PyMockAsset plug-in searches for assets under a file location specified in an environment variable called MOCK_ASSET_DIR, for example:

```
MOCK_ASSET_DIR=/tmp/MockDB
```

The entries in the selection menus in the PyMockAsset asset browser are determined by the folder structure under the location specified in the MOCK_ASSET_DIR variable.



For example, the asset browser shown above is generated from the folder structure below.



NOTE: If `MOCK_ASSET_DIR` is not set on your system, PyMockAsset defaults to searching the `/tmp` directory.

Retrieve and Publish

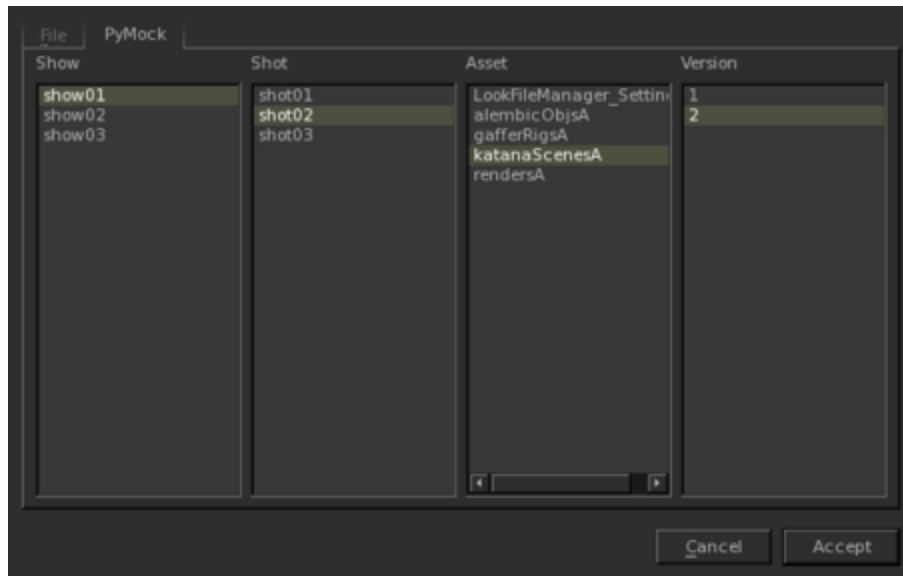
Accessing assets through the UI is performed using the asset browser provided by your plug-in. The browsers used for retrieve and publish can be different. For example, when retrieving assets PyMockAsset shows a browser that only allows selection of existing locations.

Retrieving

To retrieve an asset through the **File** menu:

1. With an asset plug-in enabled, choose **File > Open...**

The asset browser opens.



2. In the asset browser, select the asset you want to retrieve and click **Accept**.

The information you enter is resolved into an Asset ID, and that scene is loaded.

Retrieving through a supported node parameter works in the same way. For example, to bring in an asset-managed Alembic file using an Alembic_In node:

1. Add an Alembic_In node to your Katana recipe.
2. Select the Alembic_In node and press **Alt+E** to edit it.

Assuming you have an asset plug-in selected in the **Project Settings** tab, the **abcAsset** parameter shows the asset widget, and choosing **abcAsset > Browse...** opens the asset browser for your selected plug-in.

Supported Nodes and UI Locations

You can retrieve assets through the following nodes and UI locations:

- The **File** menu
To retrieve Katana scenes or macros.
- Alembic_In
- ScenegraphXML_In
- AttributeFile_In
- LookFileAssign

- LookFileGlobalsAssign
- LookFileMaterialIn
- LiveGroup
- ImageRead
- Importomatic
- PrmanGlobalSettings
- PrmanObjectSettings

Assets imported through the **ribInclude** parameter are supported.

- ArnoldGlobalSettings

Assets imported through the **assInclude** parameter are supported.

- PrimitiveCreate - for the following asset types:

- rib archive
- coordinate system sphere
- coordinate system plane

- Material

Shaders and their Args files can be asset-managed



NOTE: When you select a shader from the asset browser, the asset plug-in checks for a related asset-managed Args file. If one is not found, it looks up the Args file in the usual fashion, relative to the **.so** file location.

For more on the locations Katana searches for Args files, see the *Args Files In Shaders* chapter in the *Katana Technical Guide*.

- RenderProceduralArgs.



NOTE: When you select a render procedural from the asset browser, the asset plug-in checks for a related asset-managed Args file. If one is not found, it looks for an Args file with the same name as the render procedural **.so** file, in the same directory. For example, an **.so** file under **/tmp/proc.so** expects an Args file at **/tmp/proc.so.args**.

Publishing

When publishing through an Asset plug-in, information entered in the asset browser is used to build an Asset ID. For example, to publish from the file menu:

1. Choose **File > Save As...**
The asset browser opens.
2. In the asset browser, enter the information required to identify the asset to publish, then click **Accept**.

The asset plug-in performs the **Pre-Publish** step described in the [The Asset Publishing Process](#), and if that passes, Katana performs the **Publish** step to write the asset. Finally the **Post-Publish** step returns the Asset ID of the published asset.

Publishing from a node works in the same way. For example, to publish a light rig from a GafferThree node:

1. In a Katana scene containing a GafferThree node with a rig, select the GafferThree node and press **Alt+E** to edit it.
2. Select the rig you want to publish, right-click on the rig and choose **Export Rig**.

The **Export Rig** dialog opens for you to enter the required identification information. In the case of PyMockAsset, the browser has fields for **Show**, **Shot**, **Asset**, and **Version**.

Supported Nodes and Use Cases

You can publish assets through the following nodes and use cases:

- Render
- ImageWrite.



NOTE: When performing a Disk Render from a Render node or an ImageWrite node, the Pre-Publish and Post-Publish steps are not performed. To manually perform those steps, go to the **Parameters** tab of a Render or ImageWrite node and choose **Action > Pre-Render Publish Asset** and **Action > Post-Render Publish Asset**.

- LookFileBake
- LookFileMultiBake

Instancing

Overview

Instancing is the process of creating identical occurrences – instances – of geometry, at render time. Where a single piece of geometry is used multiple times in a scene, it can be beneficial to use instances, rather than copies to reduce the memory overhead of the scene. Rather than individual copies, where each copy has its own geometry attributes, instances use the geometry of a master – or source - location. This means that the memory overhead of an instance can be greatly reduced compared to a copy. The overhead of the instance location is limited to bookkeeping, such as maintaining transformation matrices.

Benefits of Instancing

The greatest benefits from instancing are seen when rendering many instances of complex source geometry. As mentioned in the [Overview](#), there is a minor bookkeeping overhead with each instance location, so if the source geometry is very light (such as a single primitive sphere), the benefits of instancing likely doesn't outweigh the cost of that additional bookkeeping.

The differing approaches adopted by the available renderers also have an impact on the level of benefit gained from instancing. For example, in PRMan, the REYES architecture processes buckets, and aggressively discards any geometry in completed buckets. So a dense, complex mesh, spanning multiple buckets is progressively discarded from memory as the relevant buckets are completed. If however the same mesh is the source for instances elsewhere in the scene, then that source geometry must persist in memory until the render concludes. For this reason, the number of copies, or density of geometry required for instancing to pay off may be high. One example of using REYES and getting a good level of benefit from instancing is rendering multiple instances of very complex source geometry, where each instance overlaps a single bucket. In that case, without instancing, the memory overhead of each bucket would be very large.

As raytracing does not generally free geometry, using instancing with raytracing typically delivers more obvious benefits, even with fewer instances and less dense geometry.

Instancing in PRMan

Katana does not add any instancing functionality beyond that available in PRMan, so it's important to understand the concepts and limitations of the PRMan approach. For more on instancing with PRMan, see the *Object Instancing* chapter in the *RenderMan documentation*.

PRMan defines `ObjectBegin` and `ObjectEnd` blocks containing the information defined at the instance source location. This block is copied - instanced - at each instance location, along with any attributes inherited by the original block.

Instance Types

When instancing with PRMan, Katana offers three instancing behaviors, set using the **instance.Type** attribute on the instance source location:

- **object**

Instancing using **RiInstance**, which is the default behavior if the **instance.Type** attribute is not set. This is the quickest option, and is generally used wherever possible. It is not available with PRMan prior to version 17.0, or if per-instance shader or co-shader overrides are used.

- **inline archive**

Collapses the source location to an inline archive, which is written to disk and used for all subsequent locations of that instance. Used wherever **object** instancing is not available.

- **katana**

The scene graph location of the instance source is wrapped in a Katana procedural. Used whenever a renderer agnostic instance workflow is required.

Per-Instance PrimVars

Although the purpose of instancing is to provide locations identical to the instance source, there may be occasions where you want per-instance surface properties. The three ways of achieving this with PRMan are:

- **Perform shader calculations in object space**

Object space is typically different for each instance.

- **Set user attributes per-instance**

You can set attributes of type **group** under **prmanStatements.attributes.user** and read them in your shaders using the **attribute()** shadeop.

- **Set primvars per-instance**

You can set primvars on instance locations and read them in your shaders using the **readprimvar()** shadeop.

You must set **value**, **scope**, and **inputType** for each primvar, where **value** is the value of the primvar, **scope** is the component level of the geometry, and **inputType** is the variable type (such as **string**). For example, to set a primvar named **primvarName** with the value **newTex.tx**, acting at **primitive** level, enter the following in an AttributeScript:

```
SetAttr( 'instance.arbitrary.primvarName.value' [ "newTex.tx" ] )
SetAttr( 'instance.arbitrary.primvarName.scope', ["primitive"] )
SetAttr( 'instance.arbitrary.primvarName.inputType', ["string"] )
```

See the *RenderMan documentation* for more on setting per-instance primvars.



NOTE: Due to a known limitation with PRMan, per-instance primvars must have their **scope** set to **primitive**. Settings for **scope** - such as **face**, or **vertex** - applicable to non-instance primvars are not applicable to per-instance primvars.

PRMan Example

As a simple example, create two primitives under a single group location, make the group location the instance source, and instance the group to three instance locations:

1. [Create Primitives Under a Single Group Location, then Make that the Instance Source](#)
2. [Create Instance Locations that Reference the Instance Source](#)
3. [Add Bounds to the Instance Locations and Force Expand the Instance Source](#)
4. [Render the Scene to See the Instances](#)

Create Primitives Under a Single Group Location, then Make that the Instance Source

1. Add two PrimitiveCreate nodes, and a Merge node to an empty recipe.

Connect the output of each PrimitiveCreate node to an input on the Merge node.

2. Set one PrimitiveCreate node's **type** parameter to **cube**, and the other to **cone**. Position the two primitives in the Viewer tab.
3. Change the name parameters of the PrimitiveCreate nodes to read **/root/world/geo/instanceGroup/primitiveCube** and **/root/world/geo/instanceGroup/primitiveCone** respectively.

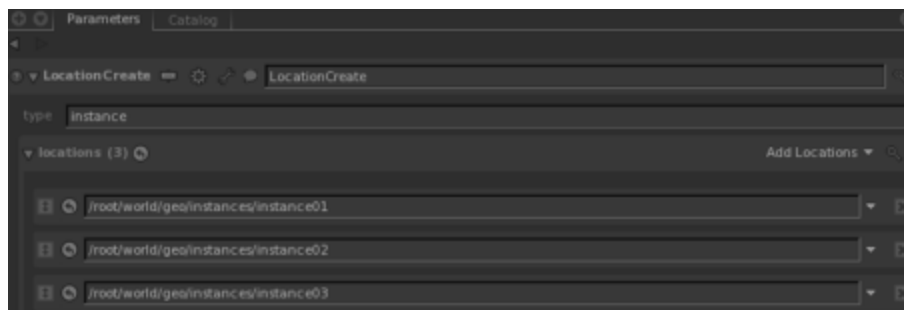


NOTE: Both primitives are grouped under a single group location, in this case named **instanceGroup**. You'll make this group location the instance source.

4. Add an AttributeSet node, downstream of the Merge node.
Specify the AttributeSet node to point to the **instanceGroup** group location in the scene graph.
5. In the AttributeSet node, set the **attributeName** parameter to **type**, the **attributeType** to **string**, and the **stringValue** to **instance source**.

Create Instance Locations that Reference the Instance Source

1. In a separate branch in the recipe, add a LocationCreate node. Edit the **type** parameter of the LocationCreate node to read **instance**.
2. Choose **Add Locations** > **path** twice, to add two more locations, and edit the paths to read **/root/world/geo/instances/instance01**, **/root/world/geo/instances/instance02**, and **/root/world/geo/instances/instance03**. This creates three locations of type instance, under a single group location (named **instances** in this case).



3. Add a Transform3D node for each instance location, then move them away from the origin, and far enough from each other that the eventual instanced geometry does not overlap.
4. Point the instance locations to the instance source.

Add an AttributeScript node, and set the location the node to run on each instance location by choosing **Add Statements** > **Custom** and entering - in this case - **/root/world/geo/instances/****

Enter the following in the AttributeScript node's **script** parameter to set each instance location's **geometry.instanceSource** attribute (in this case **/root/world/geo/instanceGroup**):

```
theSource = [ '/root/world/geo/instanceGroup' ]
SetAttr( 'geometry.instanceSource', theSource )
```

Add Bounds to the Instance Locations and Force Expand the Instance Source

You have created an instance source location, and instance locations that reference that instance source. You need to make sure that, at render time, the instance source location is expanded before any of the instances, otherwise the geometry attributes required by the instances won't be present. To do this, add bounds to each of the instance locations to make sure those locations are expanded only as needed, and force expand the instance source location to make sure it is expanded first.

1. Add an AttributeSet node downstream of the LocationCreate node that creates the instances. Point the node at the instance locations.
2. Set the AttributeScript node's **attributeName** parameter to **bound**, the **attributeType** to **double**, and the number value to a 6 X 1 array. Enter values for the bounds that you're certain encompass all of the - to be - instanced geometry. The bounds do not have to be accurate, and can be very large.



NOTE: Bounds are set using a 6 X 1 array, specifying the minimum and maximum extents of the bounding box in each of the X, Y, and Z axis. The convention is [min x, max x, min y, max y, min z, max z]

3. Add an AttributeSet node downstream of the two PrimitiveCreate nodes, and point it to the instance source location.

Set the **attributeName** parameter to **forceExpand**, the **attributeType** to **integer**, and the **numberValue** to **1.0**.

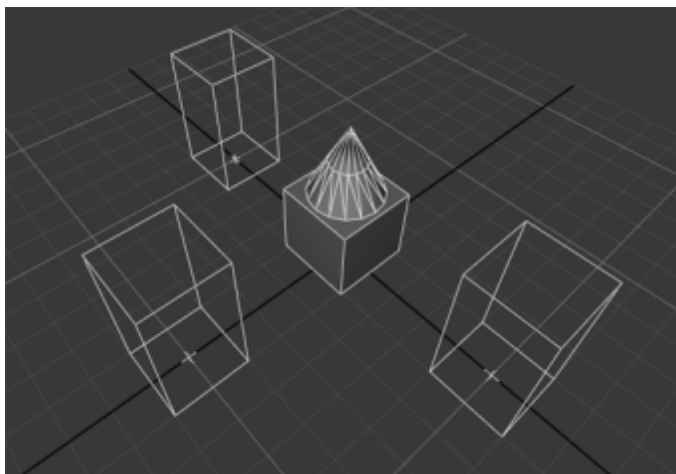
Render the Scene to See the Instances

Add a camera to your scene, so you can render, and see the instance source location instanced at each of the instance locations.

1. Add a CameraCreate node, and another Merge node, and connect the outputs of the CameraCreate node, instances branch, and instance source branch, to inputs on the Merge node. Your Node Graph should display similar to that shown below.



2. Position the camera to frame all of your instance locations in the viewer. You'll see the geometry under the instance source location, and the instance locations (represented by their bounding boxes and locators).



3. Right-click on the final Merge node and choose Preview Render. You'll see that the geometry under the instance source location is rendered, along with instances of the same geometry at each of the instance locations.



NOTE: By default, PRMan renders the instance source, and each of the instances. To prevent the instance source from rendering, add a PrmanObjectSettings node targeting the instance source location, and set the **attributes > visibility > camera** parameter to **No**.



EXPERIMENT: Create two materials and assign them to the geometry locations under the instance source location. Add a GafferThree node, and lights, then Preview Render your scene again.

Instancing in Arnold

Katana supports two types of instancing with Arnold, instancing using an instance ID and instancing from an instance source. Instancing using an instance ID is the simplest, and involves giving locations a string attribute **instance.ID**. Locations sharing the same value for **instance.ID** become instances of the first location with that ID encountered.

Instancing from an instance source operates in the same way with Arnold as it does with PRMan.

For more on instancing with Arnold, see the *Arnold documentation*.

Arnold Example

Using Instance ID

A good way to illustrate what happens when instancing with an instance ID is to create a number of primitive locations of different types (**sphere**, **cube**, **cone**, among others) give them all the same value for the **instance.ID**

attribute, then observe the results at render time:

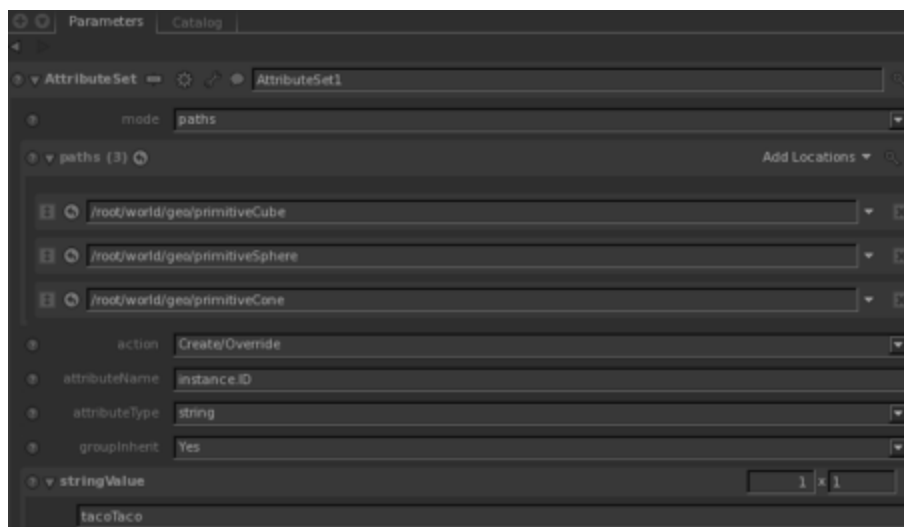
1. [Create Primitives of Different Types:](#)
2. [Give Each Primitive the Same Value for instance.ID:](#)
3. [Render Your Scene to See the Result:](#)

Create Primitives of Different Types:

1. In an empty recipe, add three PrimitiveCreate nodes, a CameraCreate node and a Merge node. Connect the outputs of each PrimitiveCreate and the CameraCreate to inputs on the Merge node.
2. Set one of the PrimitiveCreate nodes **type** parameter to **cube**, one to **sphere**, and one to **cone**, and position the primitives so that they're not overlapping.
3. In the **Viewer** tab, position the camera so that all three primitives are visible.

Give Each Primitive the Same Value for instance.ID:

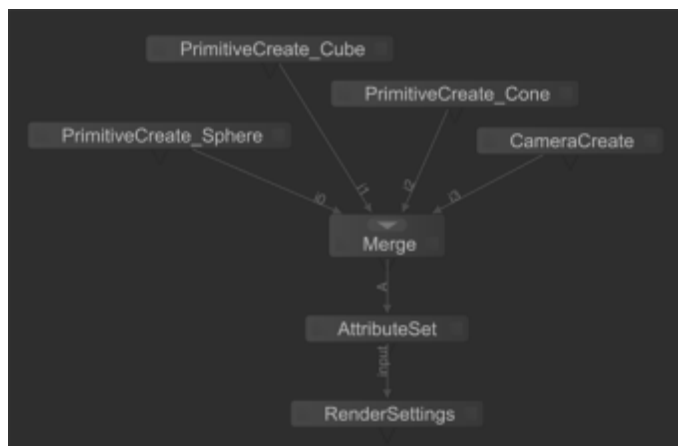
1. Add an AttributeSet node downstream of the Merge node, targeting each of the primitive locations. Set the **attributeName** parameter to **instance.ID**, the **attributeType** to **string**, and the **stringValue** to a unique identifying string.



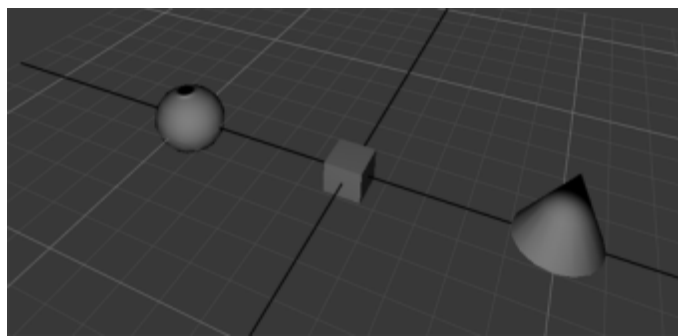
Render Your Scene to See the Result:

When the scene graph is processed, each primitive location with an instance ID takes the shape of the first location encountered with that ID.

1. Add a RenderSettings node downstream of the AttributeSet node. Set the **renderer** parameter to **arnold**.



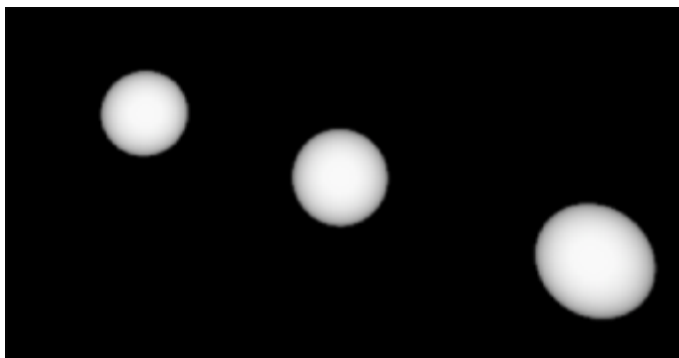
2. Right-click on the RenderSettings node and choose **Preview Render** from the context menu.



In this case, as you can see from the **Scene Graph** tab, pictured below, the primitiveSphere location is the first encountered, followed by the cube, then the cone.

Name	Type
root	group
world	group
geo	group
primitiveSphere	nurbspatch
primitiveCube	nurbspatch
primitiveCone	nurbspatch
cam	group
camera	camera
materials	group

Therefore, the cube and cone locations become instances of the sphere.



NOTE: Rather than rendering three different primitive types, each primitive location with an instance ID becomes an instance of the first location encountered with that ID.

Using Instance Source

Instancing from an instance source with Arnold, works in the same way as with PRMan. The geometry to be instanced is located under a group, with the group's type attribute set to **instance source**. The instance locations each have a **geometry.instanceSource** attribute, set to the scene graph location of the instance source. You can follow the [PRMan Example](#), adding a RenderSettings node to the recipe, with the **renderer** parameter set to **arnold**.

As with the [PRMan Example](#), it's important to force expand the instance source location, to ensure that its attributes are available when the instances are generated, and set bounds on each of the instance locations that you're sure encompass the - to be - instanced geometry.

Whether instancing using instance IDs, or instance source with Arnold, one major difference is that instancing in Arnold is per-shape, therefore each instance location can inherit material assignments, and can override inherited assignments with local material assignments per instance.



EXPERIMENT: Create three different Arnold surface shaders. Give each a different value for **Kd_color**, and assign the surface shaders to different instance locations.

Appendix A: Keyboard Shortcuts

Keyboard shortcuts provide quick access to the features of Katana. The following tables show these shortcuts.



WARNING: Currently, widget focus is not correctly restored when a dialog is opened and closed, causing certain keyboard shortcuts to stop working.

Conventions

The following conventions apply to instructions for mouse-clicks and key presses.

- LMB means click or press the left-mouse button.
- MMB means click or press the middle-mouse button
- RMB means click or press the right-mouse button.
- When you see the word “drag” after a mouse button, this tells you to press and hold the mouse button while dragging the mouse pointer.
- Keyboard shortcut combinations with the **Ctrl**, **Alt**, and **Shift** keys tell you to press and hold the key and then type the specified letter.

For example, “Press **Ctrl+S**” means hold down the **Ctrl** key, press **S**, and then release both keys.



NOTE: Keyboard shortcuts in the tables appear in upper case, but you do not type them as upper case. If the **Shift**+modifier does not appear before the letter, press the letter key alone.

Customizing Keyboard Shortcuts

Certain actions and key events in the UI can be defined with custom keyboard shortcuts using a configuration file stored in a user’s Katana folder: `$HOME/.katana/shortcuts.xml`.

Not all keyboard shortcuts can be customized. A list of modifiable keyboard shortcuts can be viewed in the **Keyboard Shortcuts** tab, along with their default assignments. Keyboard shortcuts can be customized in tabs that make use of the Keyboard Shortcuts Manager, including new custom tabs that make use of the new system.

The configuration file can be used to override the default keyboard shortcuts of actions and key events that are registered with Katana's Keyboard Shortcut Manager, for example:

```
<shortcuts>
  <shortcut id="430f81d33d338680a0c64ae9ea311cd7"
    name="SceneGraphView.ExpandBranchesToAssembly"
    shortcut="A"></shortcut>
</shortcuts>
```

The ID of a keyboard shortcut element is assigned by the user that registers the action or key event. It is a hash based on the original name of the action or key event. While the name of an action or key event may change, the ID remains the same for future versions of Katana. This ensures that the correspondence of custom keyboard shortcuts to the respective actions or key events remain the same, even if names change.



NOTE: The name attribute of a shortcut XML element only appears for readability, making it easy to identify the action or key event to which the shortcut has been assigned. The names in the shortcuts.xml file are not updated automatically when names of actions or key events are changed in the application.

You can copy an XML representation of an item in the keyboard shortcuts tree to the selection buffer clipboard by right-clicking the item and choosing **Copy as XML**. Pasting such an XML representation into the shortcuts.xml file allows you to override the custom keyboard shortcut assigned.



NOTE: In many Linux windows managers, the **Alt** key is used by default as a mouse modifier key. This can cause problems in 3D applications where **Alt** is used for camera navigation in 3D environments.

You can use key mapping to assign the mouse modifier to another key, such as the  (**Super** or **Meta**) key, but the method changes depending on which flavor of Linux you're using. Please refer to the documentation on key mapping for your particular Linux distribution for more information.

Keyboard Shortcut Manager

The `UI4.App.KeyboardShortcutManager` Python module has been added for registering action callbacks to which keyboard shortcuts can be assigned.

It covers the following areas of the UI:

- The buttons next to the main menu in Katana's main application window:
 - Shelf Scripts
 - Flush Caches
 - Toggle Scene Graph Implicit Resolvers
 - Render Only Selected Objects
 - Auto-key All
- The **Scene Graph** tab

- The table of gaffer items of GafferThree nodes that are edited in the **Parameters** tab
- Custom tab plug-ins that derive from `UI4.Tabs.BaseTab`.

General Keyboard Shortcuts

Keyboard Shortcut(s)	Action
\	Repeat the previous render.
Alt +middle-click and drag	Pans any scrollable area. (When used in the Node Graph it zooms in and out.)
Ctrl +,	Opens the Preferences dialog.
Ctrl +E	Exports the currently selected portion of the script as highlighted in the Node Graph . This saves the selected nodes as a script.
Ctrl +F	Within the Node Graph , Parameters , Scene Graph , Attributes , and Project Settings tabs, opens a Search dialog.
Ctrl +I	Imports a script into the current script.
Ctrl +MMB Drag	Allows dragging and dropping of attribute information.
Ctrl + Shift +MMB Drag	Allows dragging and dropping of attribute information.
Ctrl +N	Creates a new script. (Doesn't work inside the Node Graph .)
Ctrl +O	Opens a previously created script.
Ctrl +Q	Quit the application.
Ctrl +R	Redo the last undone action.
Ctrl +S	Save the current script.
Ctrl + Shift +S	Save the current script to a new file (Save As).
Ctrl +Z	Undo the last action.
Esc	Cancel the current render.
F4	Show shelf items.
F5	Flush caches.
F6	Toggle implicit resolvers in the Scene Graph tab.

Keyboard Shortcut(s)	Action
F7	Toggle the Render Only Selected Objects option in the Scene Graph tab.
F8	Toggle the Auto-Key All option.
P	Start an interactive render from the current view node.
Spacebar	Maximizes the pane currently below the mouse pointer. If the pane is already maximized, Spacebar restores it to its previous size.

Backdrop Node Keyboard Shortcuts

Keyboard Shortcut(s)	Action
[	Moves the Backdrop Node the mouse is over to the back.
]	Moves the Backdrop Node the mouse is over to the front.
Alt+L	Toggles the locked state of a Backdrop node.
Ctrl+LMB	Selects everything within the border of a Backdrop node.
Double LMB	Opens the Backdrop node editor.

Curve Editor Keyboard Shortcuts

Keyboard Shortcut(s)	Action
A	Frames all selected.
D	Shows domain slider.
Delete	Deletes keyframe.
Esc	Exits insert mode.
F	Frames selected.

Keyboard Shortcut(s)	Action
H	Shows heads up labels.
Insert	Enters insert mode.
S	Cycle snapping mode.

Dope Sheet Keyboard Shortcuts

Keyboard Shortcut(s)	Action
A	Frame all.
Ctrl+A	Select all.
H	Toggles tooltips.
Home	Frame global in/out.
W	Frame working in/out.

Messages Keyboard Shortcuts

Keyboard Shortcut(s)	Action
Ctrl+A	Select all.
Ctrl+C	Copy message.
Delete	Delete message.

Monitor Keyboard Shortcuts

Keyboard Shortcut(s)	Action
Full stop (.)	Toggles the pixel probe display.
Grave Accent (`)	Globally toggles the visibility of the monitor manipulators.
+/-	Zooms into and out of the image.
1-8	Switches to catalog bookmark.
1-8 (Long Press)	Sets a catalog bookmark.
Alt+K	If mask display is enabled, steps to next mask.
Alt +RMB Drag	Zooms into and out of the image.
Alt +Up Arrow/Down Arrow	Adjusts the gamma of the image.
C	Shows RGB channels as a color image.
Ctrl+Home	Resets display exposure (fstop) offset to 0.
Ctrl +LMB drag	Pixel probe.
Ctrl +Space	Maximize/minimize the Monitor tab.
Ctrl +Up Arrow/Down Arrow	Increment/decrement display exposure (fstop) offset by 1/4 stop.
Double LMB	Sets the center of 2D render spiral.
F	Fits images to Monitor tab.
Home	Reset to 1:1 zoom level, with origin in lower-left corner.
K	Toggles the mask display.
MMB drag	Pans image.
Mouse wheel	Zooms into and out of the image.
O	Toggles the overlay image on/off.

Keyboard Shortcut(s)	Action
O (press and hold)	Sets and uses the current render as the overlay image.
R/G/B/A/L	Shows red, green, blue, alpha, luminance channels as grayscale.
Shift +RMB	Toggle Region of interest (ROI) enabled state.
Shift +RMB Drag	Draws and enables Region of interest (ROI).
Space	Swaps front/back catalog display items.
Tab	Switches between monitor and catalog views.
U	Toggles the underlay image on/off.
U (press and hold)	Sets and uses the current render as the underlay image.
Up Arrow/Page Up/Down Arrow/Page Down	Views next catalog item/Views previous catalog item.

Node Graph Keyboard Shortcuts

Keyboard Shortcut(s)	Action
Full stop (.)	Adds a Dot node to the Node Graph. A Dot is only created if the mouse is over a connection, you are connecting two nodes with one end connected, or a node is selected.
Backtick (`)	Creates a connection between nodes. Press it first with the mouse over the starting node, and a second time over the node to connect to.
/	Pans the view to the node at the other end of the connection. (Only works when the mouse is hovering over one side of a connection.)
1-9	Begin or end connection with numbered output port.
A	Frames the complete node tree within the Node Graph .
Alt +Any Arrow	Nudges the selected node or nodes a small distance in the direction of the arrow.
Alt +. (full stop)	Dims the nodes that are unconnected to the viewed node.

Keyboard Shortcut(s)	Action
Alt+D	Toggles disabled state of selected nodes.
Alt+E	Opens the currently selected node's parameters settings within the Parameters tab.
Ctrl+F	Opens a Search dialog for the tab.
Alt+M	Toggles the thumbnail state of selected nodes.
Alt+G	Creates a GroupStack node with the currently selected node moved inside.
Alt+S	Toggles snapping nodes to grid while dragging within the Node Graph . When selected, moving nodes happens in steps that correspond to a grid.
Alt+T	Toggles teleport link visibility.
Ctrl+C	Copies the currently selected node or nodes to the buffer.
Ctrl+Backspace	Goes up a level from level under mouse pointer.
Ctrl+Down Arrow	Selects all nodes downstream of the currently selected node(s).
Ctrl+MMB	Enter group under mouse pointer.
Ctrl+Return	
Ctrl+Shift+Backspace	Goes up to /root level from level under mouse pointer.
Ctrl+Up Arrow	Selects all nodes upstream of the currently selected node(s).
Ctrl+V	Pastes the buffer to the Node Graph .
Ctrl+X	Deletes the currently selected node or nodes from the Node Graph and copies them to the buffer.
D	Toggles the disable state of the node currently under the mouse pointer.
Delete	Deletes the selected node from the Node Graph .
E	Opens the Parameter tab of the node currently under the mouse pointer in the Parameters tab.
Esc	Cancels whatever operation you are in the middle of, such as connecting nodes.
F	Frames the currently selected node(s) within the Node Graph .
G	Creates a Group node with the currently selected nodes moved inside.

Keyboard Shortcut(s)	Action
J	Displays the Jump-to menu which comprises all Backdrop Nodes that have the bookmark flag set. Selecting one of the menu options frames its corresponding Backdrop Node within the Node Graph .
N	Displays the right-click node creation menu at the current mouse location.
Q	Toggles showing expression links within the Node Graph . When selected, nodes that derive their parameters via an expression that references another node have this relationship shown via a black dashed line.
R	Replaces the currently selected node with a node selected from the node creation menu without disconnecting it, which is displayed when the key is pressed. Typing additional characters filters the displayed list.
Shift+~	Swaps inputs one and two on the node below the mouse pointer on a 2-input node.
T	Displays the node type of the node below the mouse pointer.
Tab	Displays the node creation menu. Typing additional characters filters the displayed list.
U	Ungroups all nodes from within the currently selected Group node and brings them into the current view, deleting the Group node.
V	Makes the currently selected node the view node for the scene graph. After pressing this key, the Scene Graph tab displays a snapshot of the scene at that point within the script.
X	Removes all connections to or from the selected node, extracting it from the node tree. Expression references to or from the node remain unchanged.
Y	Aligns selected nodes.

Parameters Keyboard Shortcuts

Keyboard Shortcut(s)	Action
D	Toggles the disabled state of node.
Ctrl+F	Opens a Search dialog for the tab.

Keyboard Shortcut(s)	Action
LMB Drag	Adjusts parameter value in Parameters tab.
Ctrl +LMB Drag	Adjusts parameter value (finer grain).
Shift +LMB Drag	Adjusts parameter value (coarser grain).

Python Console Keyboard Shortcuts

Keyboard Shortcut(s)	Action
Alt +Up Arrow	Previous history.
Alt +Down Arrow	Next history.
Ctrl +X	Cut.
Ctrl +C	Copy.
Ctrl +V	Paste.
Ctrl +Return	Execute entered script.

Timeline Keyboard Shortcuts

Keyboard Shortcut(s)	Action
[	Sets global in from current frame.
]	Sets global out from current frame.
+/-	Zooms in and out of timeline.
Alt +LMB Drag	Zooms in and out of timeline.
Ctrl +Left Arrow	Move the current frame to the previous keyframe.
Ctrl +LMB Drag	Selects a new active range.

Keyboard Shortcut(s)	Action
Ctrl+Right Arrow	Move the current frame to the next keyframe.
Home	Sets active range from global in/out.
Left Arrow/Right Arrow	Decrement the current frame by Inc. (Inc can be found on the Timeline.)/Increment the current frame by Inc. (Inc can be found on the Timeline.)
Mouse wheel	Zooms into and out of the active range.

Viewer Keyboard Shortcuts

Keyboard Shortcut(s)	Action
Grave Accent (`)	Displays the manipulators.
+/-	Adjusts the size of the manipulators.
0	Selection > Subd level 0 (base).
1	Selection > Subd level 1.
2	Selection > Subd level 2.
3	Selection > Subd level 3.
4	Displays wireframe.
5	Displays shaded (filmlook).
6	Displays shaded (simple).
7	Displays all lights.
8	Displays selected lights.
A	Toggle quick editor.
Alt+L/R/M	Mono mode, left/right/main view.
Alt+S	Stereo mode.
Backspace	Selection history backward.

Keyboard Shortcut(s)	Action
Shift +Backspace	Selection history forward.
Ctrl + B	Displays proxies bounding box.
Ctrl + G	Displays proxies geometry.
Ctrl + Shift + G	Displays both proxies bounding box and geometry.
Ctrl +LMB	Removes object from selection.
Ctrl +LMB drag	Removes objects from selection.
Ctrl + Shift +LMB	Adds object to selection.
Ctrl + Shift +LMB drag	Adds objects to selection.
C	Toggles between the perspective and orthographic views for the default persp camera.
D	Makes rotate (world) manipulators appear.
E	Makes rotate manipulators appear.
Esc	Removes all manipulators from view.
G	Displays grid.
H	Hides selection.
LMB	Selects object.
LMB Drag	Selects objects.
N	Displays normals.
P	Pins manipulator.
Q	Removes all transform manipulators from view.
R	Makes scale manipulators appear.
S	Makes translate (world) manipulators appear.
Shift + ~	Displays annotations.
T	Changes background color to black.

Keyboard Shortcut(s)	Action
Alt+T	Changes background color to gray.
Shift+T	Changes background color to white.
Tab	Displays the manipulator measurement tool.
U	Unhides selection.
W	Makes transform manipulators appear.
X, Y, or Z	Switches the camera to look along that axis.
Shift+X, Y, or Z	Switches the camera to look along that negative axis.

Appendix B: Expressions

Expressions

Katana uses Python to evaluate its expressions. Code that would evaluate as a variable assignment within Python can be used as an expression within Katana. The following appendix lists functions and variables that are specific to the expression editor and also lists which modules are imported. This is not meant to be an introduction to Python but a quick reference for those wishing to leverage some of the expression specific functions and variables.

Variables Within Expressions

Variable	Description
frame or localframe	The current frame. For example: frame - 1
globals	The project settings as shown within the Project Settings tab exposed as object parameter references. For example: globals.inTime
katanaVersion	The current Katana version - complete with build number. For instance 1.1.3
katanaBranch	The current Katana version - major and minor release numbers only. For instance 1.1
nodeName	The current node's name.

Katana Expression Functions

Function	Description
<code>getRenderLocation(<nodeObject>, <renderPass>)</code>	Returns the file asset path created by the given node object <nodeObject> for the render pass <renderPass>. Example: <pre>getRenderLocation(getNode('shadow_key'), 'primary')</pre>
<code>getNode(<nodeName>)</code>	Returns the node object with the given name <nodeName>. Example: <pre>getNode('BakeCameraCreate').fov</pre>
<code>getParam(<nodeName.param>)</code>	Returns the parameter object representing the node graph parameter referenced by its node name <nodeName> and parameter path <param>. Example: <pre>getParam("mat_blinn.shaders.surfaceParams.opacity.value")</pre>
<code>isNumber()</code>	Returns True if this parameter is a number, False otherwise.
<code>scenegraphLocationFromNode(<nodeObject>)</code>	Returns the scene graph location created by the given node object <nodeObject>. Example: <pre>scenegraphLocationFromNode(getNode('mat_katana_blinn'))</pre>

Function	Description
<code>fcurve(<fileName>, <curveName>, <frameNum>)</code>	Returns the value stored within the FCurve file <fileName> from the curve <curveName> for the given frame <frameNum>. Example: <pre>fcurve("/tmp/fcurve.xml", "lgt_spot.shaders.lightParams.intensity.value", frame)</pre> The FCurve file should be an XML file such as those generated by the menu option Export FCurve... which is obtained by right-clicking on a float parameter within the Parameters tab.
<code>getenv(<envVarName>, <defaultValue>)</code>	Returns the value of the environment variable <envVarName> or if not found the default <defaultValue>. Example: <pre>getenv("HOME", "/tmp")</pre>
<code>getres(<resolutionName>)</code>	Returns a tuple of the resolution named by <resolutionName> with the first index being the width and the second being the height. Examples: <pre>getres('hd')</pre> Returns the resolution tuple for hd in the resolution table. <pre>int(getres(globals.resolution)[0]) if int(getres(globals.resolution)[0]) > 1024 else 1024</pre> (use the width of the resolution if it is greater than 1024 otherwise use 1024)
<code>getresdict(<resolutionName>)</code>	Returns a dict of the resolution named by <resolutionName>. Example: <pre>getresdict('hd')</pre>

Function	Description
<code>getExrAttr(<fileName>, <attrHeader>, <frame>, <default = None>)</code>	Returns the string value of the attribute <attrHeader> within the file <fileName> for the given frame <frame> or the default if no attribute is found. Example: <pre>getExrAttr('/tmp/white_ncf.exr', 'spi:package', frame)</pre>



NOTE: Both **getNode()** and **getParam()** update automatically based on node name changes. When <nodeName> is changed within the **Node graph** the change is reflected within the function call. For example, If the node named MainCameraCreate within the **Node graph** were changed to BakeCamera any expressions which have the line:

```
getNode('MainCameraCreate')
```

would become:

```
getNode('BakeCamera')
```

Python Modules Within Expressions

Module	Scope	Brief Description	Module Contents
re	global	Regular expression support.	Functions: compile, escape, findall, finditer, match, purge, search, split, sub, subn, template Example: <pre>re.sub(r'x', ' by ', '2048x2048')</pre> For further help: <pre>help(re)</pre>

Module	Scope	Brief Description	Module Contents
math	expression	Standard mathematical functions and variables	Functions: acos, acosh, asin, asinh, atan, atan2, atanh, ceil, copysign, cos, cosh, degrees, exp, fabs, factorial, floor, fmod, frexp, fsum, hypot, isinf, isnan, ldexp, log, log10, log1p, modf, pow, radians, sin, sinh, sqrt, tan, tanh, trunc Variables: e, pi Example: <pre>cos(pi/3)</pre> For further help: <pre>help(math)</pre>
array	global	Efficient class based array handling	Methods: append, buffer_info, byteswap, count, extend, fromfile, fromlist, fromstring, fromunicode, index, insert, pop, read, remove, reverse, tofile, tolist, tostring, tounicode, write Example: <pre>array.array('u', "efficient").buffer_info()[1]</pre> For further help: <pre>import array help(array)</pre>

Module	Scope	Brief Description	Module Contents
os.path	as path	Common functions for manipulating pathnames	Functions: abspath, basename, commonprefix, dirname, exists, expanduser, expandvars, getatime, getctime, getmtime, getsize, isabs, isdir, isfile, islink, ismount, join, lexists, normcase, normpath, realpath, relpath, samefile, sameopenfile, samestat, split, splitdrive, splitext, walk Example: <pre>path.exists(getenv("HOME", "")) + "/shaders")</pre> For further help: <pre>help(os.path)</pre>
ExpressionMath	expression	Python interface to SPI ExpressionMath library	Functions: cfit, clamp, fit, hsvtorgb, ifelse, isfinite, isinf, isnan, lerp, matmultvec, noise, randval, retime, rgbtohsb, smoothstep, snoise, softcfit, stablehash Example: <pre>randval(0, 1, frame)</pre> For further help: <pre>help(ExpressionMath)</pre>



NOTE: To access the help for the modules type the help examples within the **Python** tab.

Appendix C: CEL and Collections

Collection Expression Language (CEL)

CEL is a grammar for specifying a sub-set of the locations within the scene graph. The locations can be selected based on their path (including wildcards), attributes, or other collections. Basic set notation is included within the grammar.

Basic CEL Syntax

CEL	Description
/root/world/cam_main	Explicitly selects the location.
/root/world/lgt*	Select all immediate children of /root/world whose name starts with lgt. Use * as a wildcard.
/root/materials//*	Selects all locations recursively beneath /root/materials . Use // to represent recursively.
//shape	Selects all locations named 'shape' anywhere within the Scene Graph tab. Use // at the start of the line to represent anywhere within the Scene Graph tab.

Value Expressions

CEL	Description
<pre>/root/world/*{ attr("type") == "camera" }</pre> OR <pre>/root/world/*{@type == "camera"}</pre>	Selects all immediate children of /root/world whose type attribute has a value of camera. Use <code>attr("<attribute>")</code> or <code>@<attribute></code> to get the value of a local attribute.
<pre>/root/world//*{ hasattr("textures.ColMap") }</pre>	Selects all locations recursively beneath /root/world which have a local attribute named textures.ColMap. Use <code>hasattr("<attribute>")</code> to check if an attribute exists on a location.
<pre>/root/world//*{ globalattr("material.surfaceParams.Ks") > 0.0 }</pre>	Selects all locations recursively beneath /root/world which have or inherit an attribute named material.surfaceParams.Ks with a value above 0.0. Use <code>globalattr("<attribute>")</code> to get the value of an attribute which is either locally assigned or inherited.
<pre>//*{@materialAssign ~= "^ambient"}</pre>	Selects all locations recursively that have the materialAssign attribute beginning with ambient. Use <code>~=</code> for regular expression syntax.

CEL Sets

CEL	Description
<pre>/root/world/lgt_key + /root/world/lgt_fill</pre> OR <pre>/root/world/lgt_key /root/world/lgt_ fill</pre>	The lgt_key and lgt_fill locations are combined (this is the default if no operator is specified). To get the union of two sets, use the + operator (or no operator).

CEL	Description
<code>/root/world/lgt* - /root/world/lgt_rim</code>	Selects all of the immediate children of /root/world that start with lgt except lgt_rim. To get the difference of two sets, use the - operator.
<code>/root/world/*a* ^ /root/world/*b*</code>	Selects all of the immediate children of /root/world that contain both an a and a b. To get the intersection of two sets, use the ^ operator.

Collections

Collections are used to store a CEL statement. Collections can also be used to make the CEL expressions local to a branch of the scene graph (or kept global under **/root**). They are stored as attributes at the location defined by the **location** parameter in the **CollectionCreate** node. As they are simply attributes within the scene graph, Collections can be included within Katana Look Files.

Collection Syntax

CEL	Description
<code>/\$my_collection/*</code>	Select the contents of the Collection in /root called my_collection. Use <code>\$<collection_name></code> to use the contents of the CEL statement.
<code>/primitive (within a local collection)</code>	For a Collection with location /root/world/geo, /primitive resolves to /root/world/geo/primitive. Use / to represent the root directory of the collection. The exception being a global collection where the full location path is needed.

CEL Pattern Extensions

CEL pattern extensions provide a way of matching children of the locations matched by a CEL expression. Commonly this is used to take a collection that already exists (for instance, root locations of certain assets) and create a CEL statement that matches specified child locations of that collection.

The CEL pattern extensions are expressions of the form **(X)<Y>**, where **X**, the left-hand side, is a CEL Set Expression and **Y**, the right-hand side, is a single CEL Pattern. There is optional whitespace surrounding the angle brackets and parentheses.

Conceptually, when evaluating expressions of this form, Katana first evaluates the set of locations that are matched by the left-hand side **X** and then concatenates the pattern **Y** onto each of these locations, with each concatenation effectively generating a new pattern. The set of locations matched by the overall expression is the union of all of these patterns.

Example CEL Pattern Extensions

CEL Pattern Extensions	Description
<code>(/root/world/cam* - /root/world/cam2)< /*{ @type == "camera" } ></code>	This matches locations of type camera beneath any location matched by <code>(/root/world/cam* - /root/world/cam2)</code>
<code>(\$cameras)< /*{ @type == "camera" } ></code>	This matches locations of type camera beneath any location matched by the root collection cameras .

There is a further restriction on the pattern on the right-hand side: it may not reference a collection. For example, expressions like **(/root)<\$collection>** or **(/root)<\$collection*>** are disallowed, and raise an error.



NOTE: The new functionality is designed to replace the undocumented **APPLY()** operator that was present in earlier versions of Katana.

Appendix D: AttributeScript to OpScript Conversion

Functions

The following Python function examples for AttributeScript have Lua function examples for use with OpScript. This is not an exhaustive list, but may offer a helpful "cheat sheet" for those with AttributeScripts in existing scenes who may wish to convert over to OpScript. For more information on OpScript, please refer to [Working with Attributes](#) on page 297, or look at the OpScript tutorials in the **Help > Example Projects** menu in Katana. For details on the OpScript API, see the *OpScript Reference* documentation available from the **Help > Documentation** menu in Katana.

GetAttr

Locally Set Values

Python:

```
tacoList = GetAttr("taco")          # local value
if tacoList:                        # returns None if not available locally
    tacoSingleValue = tacoList[0]
```

Lua:

```
-- local value
local tacoAttr = Interface.GetAttr("taco")
if tacoAttr then
    -- Convert to a table/list by calling the getNearestSample method
    local tacoList = tacoAttr:getNearestSample(0.0)
    -- First element (Note that Lua table indices start at 1)
    local tacoSingleValue = tacoList[1]
end
```

If you just want the first value at time 0.0, there's a convenient function for that in Lua:

```
-- local value
local tacoAttr = Interface.GetAttr("taco")
if tacoAttr then
    local tacoSingleValue = tacoAttr:getValue()
end
```

Inherited Values

Python:

```
# inherited value. GetAttr() returns None if not available globally or locally
tacoInheritedList = GetAttr("taco", inherit=True)
if tacoInheritedList:
    tacoInheritedValue = tacoInheritedList[0]
```

Lua:

```
-- inherited value
local tacoInheritedAttr = Interface.GetGlobalAttr("taco")
if tacoInheritedAttr then
    -- Convert to a table/list by calling getNearestSample()
    local tacoInheritedList = tacoInheritedAttr:getNearestSample(0.0)
    -- First index (Note the 1, not the 0)
    local tacoInheritedValue = tacoInheritedList[1]
end
```

If you just want the first value at time 0.0, there's a convenient method for that in OpScript:

```
-- inherited value
local tacoInheritedAttr = Interface.GetGlobalAttr("taco")
if tacoInheritedAttr then
    local tacoInheritedValue = tacoInheritedAttr:getValue()
end
```

Specifying a Location

Python:

```
tacoAttrAtLocation = GetAttr('taco', atLocation='/root/whatever')
```

Lua:

```
local tacoAttrAtLocation = Interface.GetAttr('taco', '/root/whatever')
```

Additional Inputs

With Python, AttributeScripts were limited to 1 input, so you couldn't add more inputs to a node. OpScript removes this restriction: OpScript nodes have a **Display as multi-input** option that can be enabled. The OpScript is run on each location matches by its CEL expression, as evaluated in the left-most input (usually named "i0"). If you want to read an attribute from an additional input, you can specify the scenegraph location and input index:

```
-- Read from second input (for example, "i1").
local tacoAttrFromSecondInput = Interface.GetAttr('taco', '/root/whatever', 1)
```

SetAttr

Python:

```
SetAttr("type", ["subdmesh"])
```

Lua:

```
Interface.SetAttr("type", StringAttribute("subdmesh"))
```

Unlike AttributeScript, there is no implicit conversion from primitive types like strings and numbers (or lists thereof) to their corresponding Attribute types. OpScript requires that you construct an Attribute instance explicitly.

Listed below are the constructors for the various Attribute sub-classes:

- StringAttribute(string)
- FloatAttribute(number)
- DoubleAttribute(number)
- IntAttribute(number)
- GroupAttribute()
- NullAttribute()



NOTE: To delete an attribute in Lua, use the **DeleteAttr()** function provided by the **Interface** module.

Deleting an Attribute

To delete an attribute in Lua, use the **DeleteAttr()** functions provided by the **Interface** module.

Python:

```
SetAttr("taco") # setting it to "nothing" deletes it in python
```

Lua:

```
Interface.DeleteAttr("taco")
```

GetName, GetFullName

Python:

```
name = GetName()
fullname = GetFullName()
```

Lua:

```
local name = Interface.GetOutputName()
local fullname = Interface.GetOutputLocationPath()
```

XForm Manipulation

```
local xformGroup = Interface.GetGlobalXFormGroup()
local xformMatrix = XFormUtils.CalcTransformMatrixAtTime(xformGroup, 0.0)
```

Util.AssetResolve

Python:

```
resolvedAsset = Util.AssetResolve("foo://bar")
```

Lua:

```
local assetPlugin = Asset.GetDefaultAssetPlugin()
local resolvedAsset = assetPlugin:resolveAsset("foo://bar")
```

User Parameters

Python:

```
tacoList = user.taco
tacoSingleValue = tacoList[0]
```

Lua:

```
local tacoParam = Interface.GetOpArg("user.taco")
if tacoParam then
    local tacoList = tacoParam:getNearestSample(0.0)
    local tacoSingleValue = tacoParam:getValue()
end
```

Language Basics

print()

Python:

```
print "tacoSingleValue", tacoSingleValue
print "tacoList", tacoList
```

Lua:

```
print("tacoSingleValue", tacoSingleValue)

-- if you have an 'array-like' table with consecutive indices
for i, value in ipairs(tacoList) do
    print("tacoList[" .. i .. "]", value)
end

-- if you have a 'dictionary-like' table. Note that the order of iteration is
-- undefined, much like Python

for key, value in pairs (tacoDict) do
    print("tacoDict[" .. key .. "]", value)
end
```

dir()

The **dir()** function in Python lets you introspect an object (or the current local scope), which can be useful for seeing what tools you have at your disposal. Lua doesn't have a precise equivalent, but any table can be iterated over using the `pairs()` function, so you can explore a lot in a similar way.

Python:

```
dir()
dir(some_object)
```

Lua:

```
local keys={}
for key, value in pairs(Interface) do
    table.insert(keys, key)
end
table.sort(keys)
```

```
for index, value in ipairs(keys) do
    print(value)
end
```

Comments

Python:

```
# This is a single line comment
```

Lua:

```
-- This is a single-line comment
--[This is a
    multi-line
    comment]]
```

String Concatenation

Python:

```
tacoConcat = "taco1" + "taco2"
```

Lua:

```
local tacoConcat = "taco1" .. "taco2"
```

String Manipulation

Most of the string operations you would do in Python are available in OpScript as part of the pystring module. For instance to split a string and get the last token, follow the example below.

Python:

```
path = "/foo/bar/baz"
lastToken = path.split('/')[ -1]
```

Lua, using pystring:

```
local path = "/foo/bar/baz"
local pathTokens = pystring.split(path, "/")
local lastToken = pathTokens[#pathTokens]
```

Lua, the "native" way:

```
local path = "/foo/bar/baz"
local lastToken = nil
-- string.gmatch returns an iterator of matches
for token in string.gmatch(path, "[^/]+") do
    lastToken = token
end
```

```
end
```

Slicing

Lua, the "native" way to slice a table requires a function:

```
local function slice(values, i1, i2)
    local n = #values
    -- default values for range
    i1 = i1 or 1
    i2 = i2 or n
    if i2 < 0 then
        i2 = n + i2 + 1
    elseif i2 > n then
        i2 = n
    end
    if i1 < 1 or i1 > n then
        return {}
    end

    local result = {}
    for i = i1,i2 do
        table.insert(result, values[i])
    end
    return result
end

local path = "/foo/bar/baz"
local pathTokens = pystring.split(path, "/")
pathTokens = slice(pathTokens, 1, -2)

local path = pystring.join("/", pathTokens)
```

Math

math library

The "math" library is already loaded, by default. You can just use its functions, which are mostly the same as in Python. Examples include:

- math.sqrt()
- math.abs()
- math.sin()

The official documentation of the "math" library can be found at <http://www.lua.org/manual/5.1/manual.html#5.6>.

Example usage of the "math" library can be found at <http://lua-users.org/wiki/MathLibraryTutorial>.

Arithmetic

For arithmetic, you use mostly the same operators as in Python (+, -, *, /, %).

An exception is power:

Python: `x**y`

Lua: `x^y`

You can alternately use the math library: `math.pow(x,y)`.

Executing External Scripts

Like **execfile** in AttributeScript, **dofile** in OpScript caches the compiled code for each file executed so that cooking thousands of locations on the render farm does not bring down the network.

Python:

```
execfile("/my/external/attribute_script.py")
```

Lua:

```
dofile("/my/external/op_script.lua")
```

Caveats

Be aware that module functions that directly map to existing C++ APIs use zero-based indices, even though Lua indices generally start at 1. This is done to make code as portable as possible between Lua and C++. For functions that return native Lua tables or list tables (such as **IntAttribute:nearestSample(0.0)**), the return values are indexed in the Lua standard way, starting at 1.

Lua:

```
-- getChildName index argument is zero-based
for i=0, groupAttr:getNumberOfChildren() - 1 do
    print(i .. ': ' .. groupAttr:getChildName(i))
end
```

```
-- getNearestSample returns a table that uses one-based indices, as is typical
-- in Lua.
```

```
local attr = IntAttribute({1, 2, 3})
local values = attr:nearestSample(0.0)
```

```
-- Prints "1: 1", "2: 2", "3: 3"  
for index, value in ipairs(values) do  
    print(index .. ': ' .. value)  
end
```

Appendix E: External Software

Third-Party Library Versions

The following table lists the libraries included in, or used by Katana, and their current versions.

Library	Version	Library	Version
Alembic	1.0.5 and 1.5.3	Lua	5.1.5
Arnold	4.2.4.0	Minizip	1.1
Boost	1.46.0	Netlib	1.14.0
Cg	1.3	Numpy	1.5.1
Curl	7.21.1	OpenColorIO	1.0.7
decorator	3.4.0	OpenEXR	1.6.1 and 2.0.1
Dirent	1.0	OpenImageIO	0.10
DynASM	1.3.0	OpenSceneGraph	2.8.3
Expat	2.0.1	OpenSSL	1.0.0a
Fontconfig	2.8.0	PyGraphviz	1.1
Fpconst	0.7.2	PyOpenGL	3.0.1
Freetype	2.4.4	pyparsing	1.5.3

Library	Version	Library	Version
FTGL	2.1.3	Python	2.7.3
GLEW	1.5.8	Qt	4.8.5
Glut	3.8.0	RenderMan	17.0
Graphviz	2.28.0	SIP	4.15.5
HDF	5.1.8.7	UUID	1.0
JPEG	6b	yaml-cpp	0.3.0
Libpng	1.2.25	ZeroMQ	3.2.2
Libtiff	3.9.4	Zlib	1.2.5
Log4cplus	1.0.4		



NOTE: In pre-2.0v1 versions, Katana shipped with a build of OpenColorIO that used **FnOpenColorIO** namespaced symbols, but the OpenColorIO libraries were not named accordingly (**libOpenColorIO.so** and **PyOpenColorIO.so** (Python bindings)). Due to this, using a custom facility-installed OpenColorIO in parallel with the Katana libraries encountered some problems.

This has been updated so that the libraries now shipped with Katana are Fn-prefixed too (**libFnOpenColorIO.so** and **FnPyOpenColorIO.so**). Using the Python bindings is still possible through **import FnPyOpenColorIO**, and any code using **PyOpenColorIO** needs to be updated to either use **FnPyOpenColorIO**, or a facility-installed OpenColorIO can be used instead of that shipped with Katana.

Third-Party Library Licenses

This table lists third-party libraries and their licenses.

Library	Description	License
Alembic	Geometry format library	TM & © 2010-2011 Lucasfilm Entertainment Company Ltd. or Lucasfilm Ltd. All rights reserved. Industrial Light & Magic, ILM and the Bulb and Gear design logo are all registered trademarks or service marks of Lucasfilm Ltd. © 2009-2013 Sony Pictures Imageworks Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of Industrial Light & Magic nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
Base32	Base32 implementation	Base32 implementation Copyright 2010 Google Inc. Author: Markus Gutschke Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
Boost	Source code function/template library	Boost Software License - Version 1.0 - August 17th, 2003 Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following: The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Library	Description	License
cgkit	Python Computer Graphics Kit	Copyright © 2004 Matthias Baas (baas@ira.uka.de) This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.
Curl	URL transfer library	Copyright © 1996 - 2010, Daniel Stenberg, <daniel@haxx.se>. All rights reserved. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Library	Description	License
decorator	Python 'decorator' module	Copyright © 2005-2012, Michele Simionato All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in bytecode form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Dirent	POSIX Directory Browsing API for Windows	Copyright Kevlin Henney, 1997, 2003. All rights reserved. Permission to use, copy, modify, and distribute this software and its documentation for any purpose is hereby granted without fee, provided that this copyright and permissions notice appear in all copies and derivatives. This software is supplied ""as is"" without express or implied warranty. But that said, if there are any problems please get in touch.

Library	Description	License
DSOnoises	Generation of cellular noise	Copyright © 2005 by Stefan Gustavson. All rights reserved. This code is licensed to you under the terms of the MIT license: Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ""Software""), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED ""AS IS"", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Library	Description	License
DynASM	Dynamic assembler for code generation engines	Copyright © 2005-2013 Mike Pall. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ""Software""), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED ""AS IS"", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Enum	Enumerated types for Python	Licensed under the Python licence, (a copy of which can be obtained here: https://docs.python.org/2.7/license.html#history-and-license) as per this copyright notice: Copyright © 2007–2009 Ben Finney <ben+python@benfinney.id.au> This is free software; you may copy, modify and/or distribute this work under the terms of the GNU General Public License, version 2 or later or, at your option, the terms of the Python license.

Library	Description	License
Expat	C XML Parser library	Copyright © 1998, 1999, 2000 Thai Open Source Software Center Ltd and Clark Cooper Copyright © 2001, 2002, 2003, 2004, 2005, 2006 Expat maintainers. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Library	Description	License
Fontconfig	configuring and customizing font access	Copyright © 2001, 2003 Keith Packard Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Keith Packard not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Keith Packard makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty. THE AUTHOR(S) DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL THE AUTHOR(S) BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Fpconst	Utilities for handling IEEE 754 floating point special values	Copyright: © 2003-2005 Pfizer, Licensed to PSF under a Contributor Agreement License: Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Library	Description	License
FreeType	Font support	Portions of this software are copyright © 2008 The FreeType Project (www.freetype.org). All rights reserved. The FreeType Project LICENSE, 2006-Jan-27, Copyright 1996-2002, 2006 by David Turner, Robert Wilhelm, and Werner Lemberg. Introduction The FreeType Project is distributed in several archive packages; some of them may contain, in addition to the FreeType font engine, various tools and contributions which rely on, or relate to, the FreeType Project. This license applies to all files found in such packages, and which do not fall under their own explicit license. The license affects thus the FreeType font engine, the test programs, documentation and makefiles, at the very least. This license was inspired by the BSD, Artistic, and IJG (Independent JPEG Group) licenses, which all encourage inclusion and use of free software in commercial and freeware products alike. As a consequence, its main points are that: • We don't promise that this software works. However, we will be interested in any kind of bug reports. (`as is' distribution)</li> <li>• You can use this software for whatever you want, in parts or full form, without having to pay us. ( `royalty-free' usage) • You may not pretend that you wrote this software. If you use it, or only parts of it, in a program, you must acknowledge somewhere in your documentation that you have used the FreeType code. (`credits') We specifically permit and encourage the inclusion of this software, with or without modifications, in commercial products. We disclaim all warranties covering The FreeType Project and assume no liability related to The FreeType Project. Finally, many people asked us for a preferred form for a credit/disclaimer to use in compliance with this license. We thus encourage you to use the following text: "Portions of this software are copyright © <year> The FreeType Project (www.freetype.org). All rights reserved." Please replace <year> with the value from the FreeType version you actually use.

Library	Description	License
FreeType	(continued)	Legal Terms 0. Definitions Throughout this license, the terms `package', `FreeType Project', and `FreeType archive' refer to the set of files originally distributed by the authors (David Turner, Robert Wilhelm, and Werner Lemberg) as the `FreeType Project', be they named as alpha, beta or final release. `You' refers to the licensee, or person using the project, where `using' is a generic term including compiling the project's source code as well as linking it to form a `program' or `executable'. This program is referred to as `a program using the FreeType engine'. This license applies to all files distributed in the original FreeType Project, including all source code, binaries and documentation, unless otherwise stated in the file in its original, unmodified form as distributed in the original archive. If you are unsure whether or not a particular file is covered by this license, you must contact us to verify this. The FreeType Project is copyright © 1996-2000 by David Turner, Robert Wilhelm, and Werner Lemberg. All rights reserved except as specified below. 1. No Warranty THE FREETYPE PROJECT IS PROVIDED `AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE, OF THE FREETYPE PROJECT.

Library	Description	License
FreeType	(continued)	2. Redistribution This license grants a worldwide, royalty-free, perpetual and irrevocable right and license to use, execute, perform, compile, display, copy, create derivative works of, distribute and sublicense the FreeType Project (in both source and object code forms) and derivative works thereof for any purpose; and to authorize others to exercise some or all of the rights granted herein, subject to the following conditions: • Redistribution of source code must retain this license file (`FTL.TXT') unaltered; any additions, deletions or changes to the original files must be clearly indicated in accompanying documentation. The copyright notices of the unaltered, original files must be preserved in all copies of source files. • Redistribution in binary form must provide a disclaimer that states that the software is based in part of the work of the FreeType Team, in the distribution documentation. We also encourage you to put an URL to the FreeType web page in your documentation, though this isn't mandatory. These conditions apply to any software derived from or based on the FreeType Project, not just the unmodified files. If you use our work, you must acknowledge us. However, no fee need be paid to us. 3. Advertising Neither the FreeType authors and contributors nor you shall use the name of the other for commercial, advertising, or promotional purposes without specific prior written permission. We suggest, but do not require, that you use one or more of the following phrases to refer to this software in your documentation or advertising materials: `FreeType Project', `FreeType Engine', `FreeType library', or `FreeType Distribution'. As you have not signed this license, you are not required to accept it. However, as the FreeType Project is copyrighted material, only this license, or another one contracted with the authors, grants you the right to use, distribute, and modify it. Therefore, by using, distributing, or modifying the FreeType Project, you indicate that you understand and accept all the terms of this license.

Library	Description	License
FreeType	(continued)	4. Contacts There are two mailing lists related to FreeType: • freetype@nongnu.org Discusses general use and applications of FreeType, as well as future and wanted additions to the library and distribution. If you are looking for support, start in this list if you haven't found anything to help you in the documentation. • freetype-devel@nongnu.org Discusses bugs, as well as engine internals, design issues, specific licenses, porting, etc. Our home page can be found at: http://www.freetype.org
FTGL	OpenGL font rendering library	Herewith is a license. Basically I want you to use this software and if you think this license is preventing you from doing so let me know. Copyright © 2001-3 Henry Maddocks Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Library	Description	License
GLEW	OpenGL support	The OpenGL Extension Wrangler Library Copyright © 2002-2008, Milan Ikits <milan.ikits@ieee.org> Copyright © 2002-2008, Marcelo E. Magallon <mmagallo@debian.org> Copyright © 2002, Lev Povalahev All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. The name of the author may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
GLUT	OpenGL Utility Toolkit for writing OpenGL programs	Copyright © Mark J. Kilgard, 1994, 1995, 1996, 1998. This program is freely distributable without licensing fees and is provided without guarantee or warranty expressed or implied. This program is -not- in the public domain.

Library	Description	License
GraphViz	Open source graph visualization software	Eclipse Public License - v 1.0 THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT. 1. DEFINITIONS "Contribution" means: a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and b) in the case of each subsequent Contributor: i) changes to the Program, and ii) additions to the Program; where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program. "Contributor" means any person or entity that distributes the Program. "Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program. "Program" means the Contributions distributed in accordance with this Agreement. "Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

Library	Description	License
GraphViz	(continued)	2. GRANT OF RIGHTS a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form. b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder. c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program. d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

Library	Description	License
GraphViz	(continued)	3. REQUIREMENTS A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that: a) it complies with the terms and conditions of this Agreement; and b) its license agreement: i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose; ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits; iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange. When the Program is made available in source code form: a) it must be made available under this Agreement; and b) a copy of this Agreement must be included with each copy of the Program. Contributors may not remove or alter any copyright notices contained within the Program. Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

Library	Description	License
GraphViz	(continued)	4. COMMERCIAL DISTRIBUTION Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense. For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

Library	Description	License
GraphViz	(continued)	5. NO WARRANTY EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations. 6. DISCLAIMER OF LIABILITY EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 7. GENERAL If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable. If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

Library	Description	License
GraphViz	(continued)	All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive. Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved. This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Library	Description	License
HDF5	Hierarchical Data Format Library	Copyright Notice and License Terms for HDF5 (Hierarchical Data Format 5) Software Library and Utilities HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 2006-2010 by The HDF Group. NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 1998-2006 by the Board of Trustees of the University of Illinois. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted for any purpose (including commercial purposes) provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or materials provided with the distribution. 3. In addition, redistributions of modified forms of the source or binary code must carry prominent notices stating that the original code was changed and the date of the change. 4. All publications or advertising materials mentioning features or use of this software are asked, but not required, to acknowledge that it was developed by The HDF Group and by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign and credit the contributors. 5. Neither the name of The HDF Group, the name of the University, nor the name of any Contributor may be used to endorse or promote products derived from this software without specific prior written permission from The HDF Group, the University, or the Contributor, respectively. DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE HDF GROUP AND THE CONTRIBUTORS "AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. In no event shall The HDF Group or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage.

Library	Description	License
HDF5	(continued)	Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois, Fortner Software, Unidata Program Center (netCDF), The Independent JPEG Group (JPEG), Jean-loup Gailly and Mark Adler (gzip), and Digital Equipment Corporation (DEC). Portions of HDF5 were developed with support from the Lawrence Berkeley National Laboratory (LBNL) and the United States Department of Energy under Prime Contract No. DE-AC02-05CH11231. Portions of HDF5 were developed with support from the University of California, Lawrence Livermore National Laboratory (UC LLNL). The following statement applies to those portions of the product and must be retained in any redistribution of source code, binaries, documentation, and/or accompanying materials: This work was partially produced at the University of California, Lawrence Livermore National Laboratory (UC LLNL) under contract no. W-7405-ENG-48 (Contract 48) between the U.S. Department of Energy (DOE) and The Regents of the University of California (University) for the operation of UC LLNL. DISCLAIMER: This work was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Library	Description	License
Inconsolata	Font file	Copyright (c) 2011, Raph Levien (firstname.lastname@gmail.com), Copyright (c) 2012, Cyreal (cyreal.org) This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is copied below, and is also available with a FAQ at: http://scripts.sil.org/OFL SIL OPEN FONT LICENSE Version 1.1 - 26 February 2007 PREAMBLE The goals of the Open Font License (OFL) are to stimulate worldwide development of collaborative font projects, to support the font creation efforts of academic and linguistic communities, and to provide a free and open framework in which fonts may be shared and improved in partnership with others. The OFL allows the licensed fonts to be used, studied, modified and redistributed freely as long as they are not sold by themselves. The fonts, including any derivative works, can be bundled, embedded, redistributed and/or sold with any software provided that any reserved names are not used by derivative works. The fonts and derivatives, however, cannot be released under any other type of license. The requirement for fonts to remain under this license does not apply to any document created using the fonts or their derivatives. DEFINITIONS "Font Software" refers to the set of files released by the Copyright Holder(s) under this license and clearly marked as such. This may include source files, build scripts and documentation. "Reserved Font Name" refers to any names specified as such after the copyright statement(s). "Original Version" refers to the collection of Font Software components as distributed by the Copyright Holder(s). "Modified Version" refers to any derivative made by adding to, deleting, or substituting -- in part or in whole -- any of the components of the Original Version, by changing formats or by porting the Font Software to a new environment.

Library	Description	License
Inconsolata	(continued)	"Author" refers to any designer, engineer, programmer, technical writer or other person who contributed to the Font Software. PERMISSION & CONDITIONS Permission is hereby granted, free of charge, to any person obtaining a copy of the Font Software, to use, study, copy, merge, embed, modify, redistribute, and sell modified and unmodified copies of the Font Software, subject to the following conditions: 1) Neither the Font Software nor any of its individual components, in Original or Modified Versions, may be sold by itself. 2) Original or Modified Versions of the Font Software may be bundled, redistributed and/or sold with any software, provided that each copy contains the above copyright notice and this license. These can be included either as stand-alone text files, human-readable headers or in the appropriate machine-readable metadata fields within text or binary files as long as those fields can be easily viewed by the user. 3) No Modified Version of the Font Software may use the Reserved Font Name(s) unless explicit written permission is granted by the corresponding Copyright Holder. This restriction only applies to the primary font name as presented to the users. 4) The name(s) of the Copyright Holder(s) or the Author(s) of the Font Software shall not be used to promote, endorse or advertise any Modified Version, except to acknowledge the contribution(s) of the Copyright Holder(s) and the Author(s) or with their explicit written permission. 5) The Font Software, modified or unmodified, in part or in whole, must be distributed entirely under this license, and must not be distributed under any other license. The requirement for fonts to remain under this license does not apply to any document created using the Font Software. TERMINATION This license becomes null and void if any of the above conditions are not met.

Library	Description	License
Inconsolata	(continued)	DISCLAIMER THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Library	Description	License
JPEG	C software to implement JPEG image compression and decompression	The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided ""AS IS"", and you, its user, assume the entire risk as to its quality and accuracy. This software is copyright © 1991-1998, Thomas G. Lane. All Rights Reserved except as specified below. Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions: 1. If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation. 2. If only executable code is distributed, then the accompanying documentation must state that ""this software is based in part on the work of the Independent JPEG Group"". 3. Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind. These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us. Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as ""the Independent JPEG Group's software"". We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

Library	Description	License
JPEG	(continued)	ansi2knr.c is included in this distribution by permission of L. Peter Deutsch, sole proprietor of its copyright holder, Aladdin Enterprises of Menlo Park, CA. ansi2knr.c is NOT covered by the above copyright and conditions, but instead by the usual distribution terms of the Free Software Foundation; principally, that you must include source code if you redistribute it. (See the file ansi2knr.c for full details.) However, since ansi2knr.c is not needed as part of any program generated from the IJG code, this does not limit you more than the foregoing paragraphs do. The Unix configuration script ""configure"" was produced with GNU Autoconf. It is copyright by the Free Software Foundation but is freely distributable. The same holds for its supporting scripts (config.guess, config.sub, ltconfig, ltmain.sh). Another support script, install-sh, is copyright by M.I.T. but is also freely distributable. It appears that the arithmetic coding option of the JPEG spec is covered by patents owned by IBM, AT&T, and Mitsubishi. Hence arithmetic coding cannot legally be used without obtaining one or more licenses. For this reason, support for arithmetic coding has been removed from the free JPEG software. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.) So far as we are aware, there are no patent restrictions on the remaining code. The IJG distribution formerly included code to read and write GIF files. To avoid entanglement with the Unisys LZW patent, GIF reading support has been removed altogether, and the GIF writer has been simplified to produce ""uncompressed GIFs"". This technique does not use the LZW algorithm; the resulting GIF files are larger than usual, but are readable by all standard GIF decoders. We are required to state that ""The Graphics Interchange Format© is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

Library	Description	License
Lato	Font file	Copyright (c) 2010-2014 by tyPoland Lukasz Dzedzic (team@latofonts.com) with Reserved Font Name "Lato" This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is copied below, and is also available with a FAQ at: http://scripts.sil.org/OFL SIL OPEN FONT LICENSE Version 1.1 - 26 February 2007

Library	Description	License
Lato	(continued)	PREAMBLE The goals of the Open Font License (OFL) are to stimulate worldwide development of collaborative font projects, to support the font creation efforts of academic and linguistic communities, and to provide a free and open framework in which fonts may be shared and improved in partnership with others. The OFL allows the licensed fonts to be used, studied, modified and redistributed freely as long as they are not sold by themselves. The fonts, including any derivative works, can be bundled, embedded, redistributed and/or sold with any software provided that any reserved names are not used by derivative works. The fonts and derivatives, however, cannot be released under any other type of license. The requirement for fonts to remain under this license does not apply to any document created using the fonts or their derivatives. DEFINITIONS "Font Software" refers to the set of files released by the Copyright Holder(s) under this license and clearly marked as such. This may include source files, build scripts and documentation. "Reserved Font Name" refers to any names specified as such after the copyright statement(s). "Original Version" refers to the collection of Font Software components as distributed by the Copyright Holder(s). "Modified Version" refers to any derivative made by adding to, deleting, or substituting -- in part or in whole -- any of the components of the Original Version, by changing formats or by porting the Font Software to a new environment. "Author" refers to any designer, engineer, programmer, technical writer or other person who contributed to the Font Software. PERMISSION & CONDITIONS Permission is hereby granted, free of charge, to any person obtaining a copy of the Font Software, to use, study, copy, merge, embed, modify, redistribute, and sell modified and unmodified copies of the Font Software, subject to the following conditions:

Library	Description	License
Lato	(continued)	1) Neither the Font Software nor any of its individual components, in Original or Modified Versions, may be sold by itself. 2) Original or Modified Versions of the Font Software may be bundled, redistributed and/or sold with any software, provided that each copy contains the above copyright notice and this license. These can be included either as stand-alone text files, human-readable headers or in the appropriate machine-readable metadata fields within text or binary files as long as those fields can be easily viewed by the user. 3) No Modified Version of the Font Software may use the Reserved Font Name(s) unless explicit written permission is granted by the corresponding Copyright Holder. This restriction only applies to the primary font name as presented to the users. 4) The name(s) of the Copyright Holder(s) or the Author(s) of the Font Software shall not be used to promote, endorse or advertise any Modified Version, except to acknowledge the contribution(s) of the Copyright Holder(s) and the Author(s) or with their explicit written permission. 5) The Font Software, modified or unmodified, in part or in whole, must be distributed entirely under this license, and must not be distributed under any other license. The requirement for fonts to remain under this license does not apply to any document created using the Font Software. TERMINATION This license becomes null and void if any of the above conditions are not met. DISCLAIMER THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Library	Description	License
Libpng	PNG reference library	If you modify libpng you may insert additional notices immediately following this sentence. libpng versions 1.2.6, August 15, 2004, through 1.2.25, February 18, 2008, are Copyright © 2004, 2006-2008 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-1.2.5 with the following individual added to the list of Contributing Authors • Cosmin Truta libpng versions 1.0.7, July 1, 2000, through 1.2.5 - October 3, 2002, are Copyright © 2000-2002 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-1.0.6 with the following individuals added to the list of Contributing Authors • Simon-Pierre Cadieux • Eric S. Raymond • Gilles Vollant and with the following additions to the disclaimer: There is no warranty against interference with your enjoyment of the library or against infringement. There is no warranty that our efforts or the library will fulfill any of your particular purposes or needs. This library is provided with all faults, and the entire risk of satisfactory quality, performance, accuracy, and effort is with the user. libpng versions 0.97, January 1998, through 1.0.6, March 20, 2000, are Copyright © 1998, 1999 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-0.96, with the following individuals added to the list of Contributing Authors: • Tom Lane • Glenn Randers-Pehrson • Willem van Schaik libpng versions 0.89, June 1996, through 0.96, May 1997, are Copyright © 1996, 1997 Andreas Dilger

Library	Description	License
Libpng	(continued)	Distributed according to the same disclaimer and license as libpng-0.88, with the following individuals added to the list of Contributing Authors: • John Bowler • Kevin Bracey • Sam Bushell • Magnus Holmgren • Greg Roelofs • Tom Tanner libpng versions 0.5, May 1995, through 0.88, January 1996, are Copyright © 1995, 1996 Guy Eric Schalnat, Group 42, Inc. For the purposes of this copyright and license, "Contributing Authors" is defined as the following set of individuals: • Andreas Dilger • Dave Martindale • Guy Eric Schalnat • Paul Schmidt • Tim Wegner The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage. Permission is hereby granted to use, copy, modify, and distribute this source code, or portions hereof, for any purpose, without fee, subject to the following restrictions:

Library	Description	License
Libtiff	TIFF reference library	Copyright © 1988-1997 Sam Leffler Copyright © 1991-1997 Silicon Graphics, Inc. Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics. THE SOFTWARE IS PROVIDED ""AS-IS"" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Library	Description	License
log4cplus	C++ Logging library	Two clause BSD license: Copyright © 1999-2009 Contributors to log4cplus project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
Lua	The Lua scripting language	Copyright © 1994–2014 Lua.org, PUC-Rio. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Library	Description	License
LuaJIT	A just-in-time compiler for Lua	LuaJIT -- a Just-In-Time Compiler for Lua. http://luajit.org/ Copyright © 2005-2013 Mike Pall. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. [MIT license: http://www.opensource.org/licenses/mit-license.php] [LuaJIT includes code from Lua 5.1/5.2, which has this license statement:] Copyright © 1994-2012 Lua.org, PUC-Rio. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Library	Description	License
LuaJIT	(continued)	THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. [LuaJIT includes code from dmalloc, which has this license statement:] This is a version (aka dmalloc) of malloc/free/realloc written by Doug Lea and released to the public domain, as explained at http://creativecommons.org/licenses/publicdomain
Minizip	Library to deflate compressed files and create gzip (.gz) files	Condition of use and distribution are the same than zlib: This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution.

Library	Description	License
Modernizr	Browser feature detection library	Copyright (c) 2009–2015 Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Library	Description	License
Netlib	Repository for mathematical software, papers, and databases	Copyright © 2000-2003 TargetJr Consortium GE Corporate Research and Development (GE CRD) 1 Research Circle Niskayuna, NY 12309 All Rights Reserved Reproduction rights limited as described below. Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notice and this permission notice appear in all copies of the software and related documentation, (ii) the name TargetJr Consortium (represented by GE CRD), may not be used in any advertising or publicity relating to the software without the specific, prior written permission of GE CRD, and (iii) any modifications are clearly marked and summarized in a change history log. THE SOFTWARE IS PROVIDED ""AS IS"" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE TARGETJR CONSORTIUM BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR ON ANY THEORY OF LIABILITY ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Library	Description	License
Numpy	Linear Algebra library for Python	Copyright © 2005-2009, NumPy Developers. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of the NumPy Developers nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
NVIDIA Cg Library	High-level shading language	Copyright © 2002, NVIDIA Corporation. NVIDIA Corporation("NVIDIA") supplies this software to you in consideration of your agreement to the following terms, and your use, installation, modification or redistribution of this NVIDIA software constitutes acceptance of these terms. If you do not agree with these terms, please do not use, install, modify or redistribute this NVIDIA software. In consideration of your agreement to abide by the following terms, and subject to these terms, NVIDIA grants you a personal, non-exclusive license, under NVIDIA's copyrights in this original NVIDIA software (the "NVIDIA Software"), to use, reproduce, modify and redistribute the NVIDIA Software, with or without modifications, in source and/or binary forms; provided that if you redistribute the NVIDIA Software, you must retain the copyright notice of NVIDIA, this notice and the following text and disclaimers in all such redistributions of the NVIDIA Software. Neither the name, trademarks, service marks nor logos of NVIDIA Corporation may be used to endorse or promote products derived from the NVIDIA Software without specific prior written permission from NVIDIA. Except as expressly stated in this notice, no other rights or licenses express or implied, are granted by NVIDIA herein, including but not limited to any patent rights that may be infringed by your derivative works or by other works in which the NVIDIA Software may be incorporated. No hardware is licensed hereunder. THE NVIDIA SOFTWARE IS BEING PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR ITS USE AND OPERATION EITHER ALONE OR IN COMBINATION WITH OTHER PRODUCTS. IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, EXEMPLARY, CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, LOST PROFITS; PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) OR ARISING IN ANY WAY OUT OF THE USE, REPRODUCTION, MODIFICATION AND/OR DISTRIBUTION OF THE NVIDIA SOFTWARE, HOWEVER CAUSED AND WHETHER UNDER THEORY OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY OR OTHERWISE, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
OpenColorIO	Color management library	Copyright © 2003-2010 Sony Pictures Imageworks Inc., et al. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of Sony Pictures Imageworks nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
OpenEXR	Image file format library	Copyright © 2006, Industrial Light & Magic, a division of Lucasfilm Entertainment Company Ltd. Portions contributed and copyright held by others as indicated. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of Industrial Light & Magic nor the names of any other contributors to this software may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
OpenImageIO	Library for reading and writing images	Copyright 2008 Larry Gritz and the other authors and contributors. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of the software's owners nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. (This is the Modified BSD License)

Library	Description	License
OpenSans	font file	Copyright 2010 Google Inc. Designed by: Steve Matteson Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
Open SceneGraph	3D graphics toolkit	Copyright © 2002 Robert Osfield. Everyone is permitted to copy and distribute verbatim copies of this licence document, but changing it is not allowed. OPENSCENEGAPH PUBLIC LICENCE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION This library is free software; you can redistribute it and/or modify it under the terms of the OpenSceneGraph Public License (OSGPL) version 0.0 or later. The OSGPL is based on the LGPL, with the 4 exceptions laid out in the wxWindows section below. The LGPL is contained in the final section of this license.

Library	Description	License
OpenSSL	Toolkit that implements SSL	Copyright © 1998-2008 The OpenSSL Project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)" 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. 5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. 6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)" THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
OpenSSL	(continued)	This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Original SSLeay License Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved. This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL. This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com). Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)" The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

Library	Description	License
OpenSSL	(continued)	4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)" If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. The licence and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Library	Description	License
Ptex	Ptex library	Copyright 2009 Disney Enterprises, Inc. All rights reserved Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. The names "Disney", "Walt Disney Pictures", "Walt Disney Animation Studios" or the names of its contributors may NOT be used to endorse or promote products derived from this software without specific prior written permission from Walt Disney Pictures. Disclaimer: THIS SOFTWARE IS PROVIDED BY WALT DISNEY PICTURES AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT AND TITLE ARE DISCLAIMED. IN NO EVENT SHALL WALT DISNEY PICTURES, THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND BASED ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Library	Description	License
PyGraphviz	Python interface to Graphviz	Copyright © 2004-2010 by Aric Hagberg <hagberg@lanl.gov> Dan Schult <dschult@colgate.edu> Manos Renieris, http://www.cs.brown.edu/~er/ Distributed with BSD license. All rights reserved, see LICENSE for details. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
PyOpenGL	Standard OpenGL bindings for Python	NOTE: THIS SOFTWARE IS NOT FAULT TOLERANT AND SHOULD NOT BE USED IN ANY SITUATION ENDANGERING HUMAN LIFE OR PROPERTY. OpenGL-ctypes License Copyright © 2005-2009, Michael C. Fletcher and Contributors. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • The name of Michael C. Fletcher, or the name of any Contributor, may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS AND CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. OpenGL-ctypes includes code from the PyOpenGL 2.x series licensed under version 3 of the PyOpenGL License (BSD-style): PyOpenGL License (v3)

Library	Description	License
PyOpenGL	(continued)	PyOpenGL is based on PyOpenGL 1.5.5, Copyright &copy; 1997-1998 by James Hugunin, Cambridge MA, USA, Thomas Schwaller, Munich, Germany and David Ascher, San Francisco CA, USA. Contributors to the PyOpenGL project in addition to those listed above include: • David Konerding • Soren Renner • Rene Liebscher • Randall Hopper • Michael Fletcher • Thomas Malik • Thomas Hamelryck • Jack Jansen • Michel Sanner • Tarn Weisner Burton • Andrew Cox • Rene Dudfield PyOpenGL is Copyright © 1997-1998, 2000-2006 by the contributors. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • The names of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

Library	Description	License
PyOpenGL	(continued)	THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ""AS IS"" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. OpenGL-ctypes includes code from the Pyglet project, licensed under the Pyglet License (BSD Style): Copyright © 2006-2008 Alex Holkner. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • Neither the name of pyglet nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ""AS IS"" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

Library	Description	License
PyOpenGL	(continued)	IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. OpenGL-ctypes may include source code from the GLE (GL Tubing and Extrusion) library, which is licensed under the license declared in OpenGL/DLLS/gle_COPYING.src if GLE is included in this distribution. Copyright notice follows: This software is owned by International Business Machines Corporation ("IBM"), or its subsidiaries or IBM's suppliers, and is copyrighted and licensed, not sold. IBM retains title to the software, and grants you a nonexclusive license for the software. OpenGL-ctypes may include source code from the GLUT (GL Utility Toolkit) library, which is licensed under the following license/copyright notice (available in OpenGL/DLLS/glut_README.txt if GLUT is included in this distribution: The OpenGL Utility Toolkit distribution for Win32 (Windows NT & Windows 95) contains source code modified from the original source code for GLUT version 3.3 which was developed by Mark J. Kilgard. The original source code for GLUT is Copyright 1997 by Mark J. Kilgard. GLUT for Win32 is Copyright 1997 by Nate Robins and is not in the public domain, but it is freely distributable without licensing fees. It is provided without guarantee or warranty expressed or implied. It was ported with the permission of Mark J. Kilgard by Nate Robins. THIS SOURCE CODE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Library	Description	License
PyOpenGL	(continued)	OpenGL (R) is a registered trademark of Silicon Graphics, Inc. OpenGL-ctypes may include binary distributions of the Tk Togl widget, which is licensed under the following license/copyright notice (available in OpenGL/Tk/*/LICENSE if Togl is included). This software is copyrighted by Brian Paul (brian@mesa3d.org), Benjamin Bederson (bederson@cs.umd.edu), and Greg Couch (gregcouch@users.sourceforge.net). The following terms apply to all files associated with the software unless explicitly disclaimed in individual files. The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply. IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Library	Description	License
pyparsing	Python parsing module	Copyright © 2003-2009 Paul T. McGuire Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ""Software""), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED ""AS IS"", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Python	Programming language	Copyright © 2001, 2002, 2003, 2004 Python Software Foundation; All Rights Reserved. Licensed under the PSF license for Python 2.7. For more detail refer to https://docs.python.org/2/license.html.

Library	Description	License
Qt	Cross-platform application and UI framework for developers using C++ or QML	Digia Qt LGPL Exception version 1.1 As an additional permission to the GNU Lesser General Public License version 2.1, the object code form of a "work that uses the Library" may incorporate material from a header file that is part of the Library. You may distribute such object code under terms of your choice, provided that: (i) the header files of the Library have not been modified; and (ii) the incorporated material is limited to numerical parameters, data structure layouts, accessors, macros, inline functions and templates; and (iii) you comply with the terms of Section 6 of the GNU Lesser General Public License version 2.1. Moreover, you may apply this exception to a modified version of the Library, provided that such modification does not involve copying material from the Library into the modified Library's header files unless such material is limited to (i) numerical parameters; (ii) data structure layouts; (iii) accessors; and (iv) small macros, templates and inline functions of five lines or less in length. Furthermore, you are not required to apply this additional permission to a modified version of the Library.
Roboto Slab	Font file	Apache License Version 2.0, January 2004 http://www.apache.org/licenses/ TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION 1. Definitions. "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document. "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

Library	Description	License
Roboto Slab	(continued)	"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity. "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License. "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files. "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types. "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below). "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

Library	Description	License
Roboto Slab	(continued)	"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution." "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work. 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form. 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

Library	Description	License
Roboto Slab	(continued)	4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions: (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and (b) You must cause any modified files to carry prominent notices stating that You changed the files; and (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

Library	Description	License
Roboto Slab	(continued)	5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions. 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file. 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License. 8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

Library	Description	License
Roboto Slab	(continued)	9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability. END OF TERMS AND CONDITIONS APPENDIX: How to apply the Apache License to your work. To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives. Copyright [yyyy] [name of copyright owner] Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Library	Description	License
SIP	Creates Python bindings for C and C++ libraries	RIVERBANK COMPUTING LIMITED LICENSE AGREEMENT FOR SIP 1. This LICENSE AGREEMENT is between Riverbank Computing Limited ("Riverbank"), and the Individual or Organization ("Licensee") accessing and otherwise using SIP software in source or binary form and its associated documentation. SIP comprises a software tool for generating Python bindings for software C and C++ libraries, and a Python extension module used at runtime by those generated bindings. 2. Subject to the terms and conditions of this License Agreement, Riverbank hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use SIP alone or in any derivative version, provided, however, that Riverbank's License Agreement and Riverbank's notice of copyright, e.g., "Copyright © 2014 Riverbank Computing Limited; All Rights Reserved" are retained in SIP alone or in any derivative version prepared by Licensee. 3. In the event Licensee prepares a derivative work that is based on or incorporates SIP or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to SIP. 4. Licensee may not use SIP to generate Python bindings for any C or C++ library for which bindings are already provided by Riverbank. 5. Riverbank is making SIP available to Licensee on an "AS IS" basis. RIVERBANK MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, RIVERBANK MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF SIP WILL NOT INFRINGE ANY THIRD PARTY RIGHTS. 6. RIVERBANK SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF SIP FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING SIP, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

Library	Description	License
SIP	(continued)	7. This License Agreement will automatically terminate upon a material breach of its terms and conditions. 8. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between Riverbank and Licensee. This License Agreement does not grant permission to use Riverbank trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party. 9. By copying, installing or otherwise using SIP, Licensee agrees to be bound by the terms and conditions of this License Agreement.
UUID	Generates universally unique identifiers	Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, and the entire permission notice in its entirety, including the disclaimer of warranties. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ALL OF WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF NOT ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Library	Description	License
yaml-cpp	A YAML parser and emitter for C++	Copyright © 2008 Jesse Beder. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
ZeroMQ	An asynchronous messaging library.	Copyright © 2007-2014 iMatix Corporation and Contributors. ZeroMQ is licensed under the GNU Lesser General Public License V3 plus a static linking exception.

Library	Description	License
Zlib	Compression library	General purpose compression library version 1.2.2, October 3rd, 2004 Copyright © 1995-2004 Jean-loup Gailly and Mark Adler This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution. Jean-loup Gailly jloup@gzip.org Mark Adler madler@alumni.caltech.edu

GNU Lesser General Public License (LGPL) v2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary

GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy. This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays

copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will

automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version

or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990

Ty Coon, President of Vice

That's all there is to it!

GNU Lesser General Public License (LGPL) v3.0

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
 - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General

Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

GNU General Public License (GPL) v3.0

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps:

(1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under

this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program. You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section

7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are

not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any

covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients.

"Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

Also add information on how to contact you by electronic and paper mail. If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary.

For more information on this, and how to apply and follow the GNU GPL, see <<http://www.gnu.org/licenses/>>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <<http://www.gnu.org/philosophy/why-not-lgpl.html>>.

OpenSceneGraph v0.0

OpenSceneGraph Public License, Version 0.0

=====

Copyright (C) 2002 Robert Osfield.

Everyone is permitted to copy and distribute verbatim copies of this licence document, but changing it is not allowed.

OPENSCENEGRAPH PUBLIC LICENCE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

This library is free software; you can redistribute it and/or modify it under the terms of the OpenSceneGraph Public License (OSGPL) version 0.0 or later.

Notes: the OSGPL is based on the LGPL, with the 4 exceptions laid out in the wxWindows section below. The LGPL is contained in the final section of this license.

wxWindows Library Licence, Version 3

=====

Copyright (C) 1998 Julian Smart, Robert Roebling [, ...]

Everyone is permitted to copy and distribute verbatim copies of this licence document, but changing it is not allowed.

WXWINDOWS LIBRARY LICENCE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public Licence as published by the Free Software Foundation; either version 2 of the Licence, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public Licence for more details.

You should have received a copy of the GNU Library General Public Licence along with this software, usually in a file named COPYING.LIB. If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

EXCEPTION NOTICE

1. As a special exception, the copyright holders of this library give permission for additional uses of the text contained in this release of the library as licenced under the wxWindows Library Licence, applying either version 3 of the Licence, or (at your option) any later version of the Licence as published by the copyright holders of version 3 of the Licence document.

2. The exception is that you may use, copy, link, modify and distribute under the user's own terms, binary object code versions of works based on the Library.
3. If you copy code from files distributed under the terms of the GNU General Public Licence or the GNU Library General Public Licence into a copy of this library, as this licence permits, the exception does not apply to the code that you add in this way. To avoid misleading anyone as to the status of such modified files, you must delete this exception notice from such code and/or adjust the licensing conditions notice accordingly.
4. If you write modifications of your own for this library, it is your choice whether to permit this exception to apply to your modifications. If you do not wish that, you must delete the exception notice from such code and/or adjust the licensing conditions notice accordingly.

For the GNU LGPLv2.1, refer to [GNU Lesser General Public License \(LGPL\) v2.1](#).

Appendix F: External File Formats

Args Files

Args Files In Shaders

Args files provide additional UI hints about how parameters are to be displayed in the user interface. One of the main uses is to describe how shader parameters are presented. Various aspects, like options for a shader parameter's widget, conditional options, as well as help text, can be modified interactively inside Katana's UI. Other results, such as grouping parameters together into pages, can be achieved by editing the **.args** file directly.

When loading a shader, by default, Katana looks in ../doc relative to the directory the shader is in for the associated **.args** file.

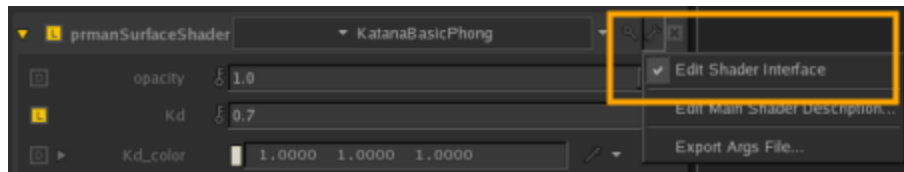
The corresponding Args file for KatanaBasicPhong reads as follows:

```
<args format="1.0">
<param name="opacity"/>
<param name="Kd"/>
<param name="Kd_color" widget="color"/>
<param name="Ks"/>
<param name="Ks_color" widget="color"/>
<param name="SpecularExponent"/>
<param name="Ka"/>
  <param name="Ka_color" widget="color"/>
</args>
```

Edit Shader Interface Interactively in the UI

Enabling Editing the User Interface

To enable editing of the shader interface, enable **Edit Shader Interface** using the wrench  button (found on the right-hand side of each shader in the Material node). The wrench button displays to the right of every parameter, allowing the configuration of the parameter's widget and help text.



Edit Main Shader Description

By selecting **Edit Main Shader Description...** from the wrench dropdown menu, a context help of the shader can be added. This description can include HTML and is shown when clicking the help  icon next to the shader name.

Export Args File

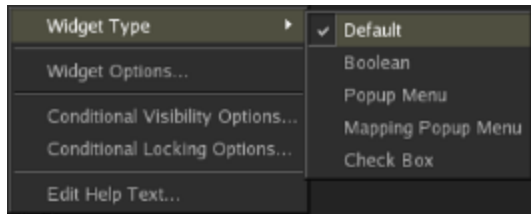
The shader interface can be exported using the **Export Args File...** command in the wrench dropdown menu.



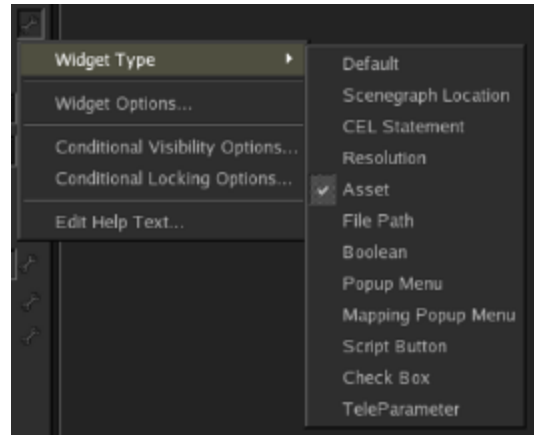
NOTE: By default, the file is saved to the /tmp directory, but other directories can also be specified.

Widget Types

Depending on the user parameter defined in a shader's Args File, different Widget Types are available to choose from. The main user parameters are the **Number**, **String**, and color parameters. The widget types available for a **Number** and a **String** shader parameter are shown below.



Number shader parameter



String shader parameter

Widget Type	Description and Example
Default	If no specific widget is set in the .args file, Katana uses the type specified in the shader and shows a widget for it automatically (for example, color picker for shader parameters of type color).
Number, String, Button, Toolbar, TeleParameter, and Node Drop Proxy	
Boolean	The field is either Yes or No . This has the same functionality as a Check Box, but displays the options as a list. <pre><param name="RepeatS" widget="boolean"/></pre>
Popup Menu	An option can be selected from a pre-defined dropdown list. See Widget Options for more information. <pre><param name="opacity" widget="popup"> <hintlist name="options"> <string value="0.0"/> <string value="0.5"/> <string value="1.0"/> </hintlist> </param></pre>

Widget Type	Description and Example
Mapping Popup Menu	Similar to Popup Menu, but with the option to map values. See Widget Options for more information. <pre><param name="opacity" widget="mapper"> <hintdict name="options"> <float value="0.0" name="A"/> <float value="0.5" name="B"/> <float value="1.0" name="C"/> </hintdict> </param></pre>
Check Box	Similar to Boolean, but displayed as a check box. <pre><param name="RepeatS" widget="checkBox"/></pre>
String, Button, Toolbar, TeleParameter, and Node Drop Proxy	
Scenegraph Location	Widget for specifying locations in the scene graph, for example, /root/world/geo/pony1. <pre><param name="loc" widget="scenegraphLocation"/></pre>
CEL Statement	Specify a CEL Statement. <pre><param name="loc" widget="cel"/></pre>
Resolution	A resolution, for exaple, 1024x768. <pre><param name="loc" widget="resolution"/></pre>
Asset	Widget to represent an asset. The fields that are displayed in the UI and the browser that is used for selection can be customized using the Asset Management System API. <pre><param name="EnvMap" widget="assetIdInput"/></pre>
File Path	String parameter representing a file on disk. Uses the standard Katana file browser for selection. <pre><param name="texname" widget="fileInput"/></pre>
Script Button	A button executing a Python script when clicked. <pre><param scriptText="print 'Hello'" name="btn" buttonText="Run Script" widget="scriptButton"/></pre>

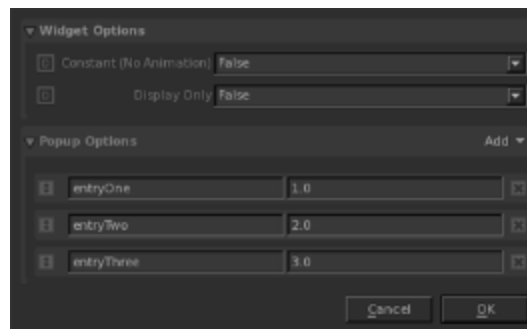
Widget Type	Description and Example
TeleParameter	A button executing a Python script when clicked. <pre><param scriptText="print 'Hello'" name="btn" buttonText="Run Script" widget="scriptButton"/></pre>
Script Editor	Creates a parameter that 'teleports' parameters from another source (node, SuperTool, or similar). <pre><param name="EnvMap" widget="teleparam"/></pre>
Dynamic Array	A number or string array of dynamic size. Not available through the UI wrench menu. <pre><numberarray_parameter hints="{&apos;widget&apos;;: &apos; dynamicArray&apos;}" name="testNumArray" size="3" tupleSize="1"> <number_parameter name="i0" value="0"/> <number_parameter name="i1" value="0"/> <number_parameter name="i2" value="0"/> </numberarray_parameter></pre>
Multi-line Text	Enables a string field to support multiple lines of text. For example, you can set KatanaBlinn.args with the following line: <pre><param name="BumpMap" widget="text"/></pre> to set BumpMap to take multiple lines of text and display the expected UI. Not available through the UI wrench menu.
Group Only	
Multi	Creates a group set of parameters within a group.
Number Array Only	
Color	The standard Katana Color picker. <pre><param name="Kd_color" widget="color"/></pre>
String Array Only	
Scene Graph Locations	Creates three Scene Graph Locations widgets that allow you to set locations.



NOTE: For more information about user parameters and widget types, refer to the *User Parameters and Widget Types* chapter in the *Katana Technical Guide*.

Widget Options

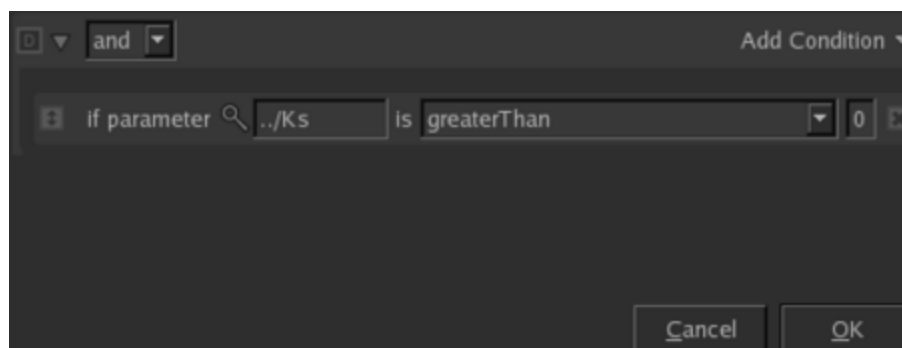
Based on the specified widget type, there are a number of options available. In the case of color parameters, for example, these options allow settings like the restriction of the components (RGBA) to a range between 0 and 1. For numeric parameters, the display format and slider options, such as range and sensitivity, can be specified.



NOTE: For more information about user parameters and widget options, refer to the [User Parameters and Widget Types](#) on page 281.

Conditional Visibility Options

Some shader parameters are not applicable or do not make sense under certain conditions. To hide these parameters from the UI, select **Conditional Visibility Options...** from the wrench dropdown menu. Multiple conditions can be matched and combined using **AND** or **OR** keywords.



This might look as follows In an **.args** file:

```
<param name="SpecularExponent">
<hintdict name="conditionalVisOps">
<string value="greaterThan" name="conditionalVisOp"/>
```

```

<string value="../../../Ks" name="conditionalVisPath"/>
<string value="0" name="conditionalVisValue"/>
</hintdict>
</param>

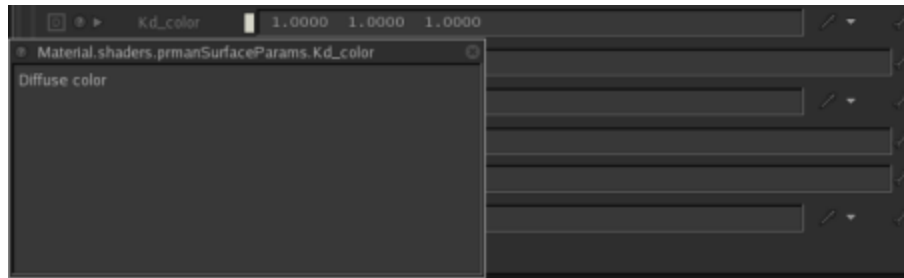
```

Conditional Locking Options

Conditional Locking works exactly like the [Conditional Visibility Options](#), with the difference of locking a parameter under the specified conditions rather than hiding it.

Editing Help Text

Similar to the Main Shader Description, an HTML help text can be specified for every parameter. The text can be specified using the **Edit Help Text...** option from the wrench dropdown menu.



In the **.args** file, this tooltip is reflected as follows:

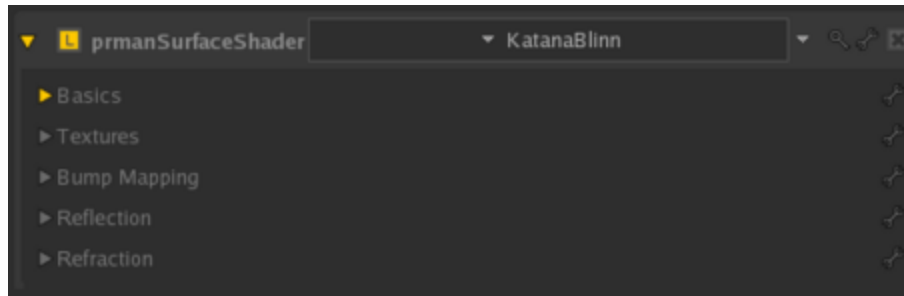
```

<args format="1.0">
<param name="Kd_color" widget="color">
<help>
Diffuse color
</help>
</param>
</args>

```

Grouping Parameters into Pages

Pages allow parameters to be grouped together to be displayed in a more organized way.



This can be achieved in two ways when editing the **.args** files:

1. Adding a page attribute with the name of the page to each parameter:

```
<param name="Kd" page="myPage" />
```

2. Grouping parameters using a page tag:

```
<page name="myPage">
```

```
.
```

```
.
```

```
</page>
```



TIP: The attribute `open` can be set to **True** to expand a group, by default. The `open` hint also works for group parameters and attributes, which are closed by default. For example:

```
<page name="Basics" open="True">
```



NOTE: Currently, when using pages, only attributes listed in a page are visible! Any parameter not specifically assigned to a page is hidden.

Co-Shaders

It is conventional in RenderMan to specify co-shaders used in a shader interface using parameters of type string or shader. If it is of type shader, Katana detects that this is a co-shader port automatically. If it is of type string, we need to provide a hint in the **.args** file.

```
<param name="Kd_color_mycoshader" coshaderPort="True" />
```

Co-Shader Pairing

Katana allows co-shaders to be represented as network materials. For user convenience, there is a convention that allows pairs of parameters, representing a value and a port for a co-shader, to be presented to the user to look like a single connectable value in the UI.

RenderMan co-shader pairing can be used by adding a co-shader port and specifying the co-shader's name in the `coshaderPair` attribute of the parameter. In the **.args** file, this can be achieved as follows:

```
<args format="1.0">
<param name="Kd_color_mycoshader" coshaderPort="True" />
<param name="Kd_color"
    coshaderPair="Kd_color_mycoshader" widget="color"/>
</args>
```

Example of an Args File

An example of an **.args** file using pages and different types of widgets is `KatanaBlinn`, saved in `SHADERS/doc`:

```
<args format="1.0">
<page name="Basics" open="True">
    <param name="Kd"/>
<param name="Kd_color" widget="color"/>
<param name="Ks"/>
<param name="Ks_color" widget="color"/>
<param name="Roughness"/>
<param name="Ka"/>
    <param name="Ka_color" widget="color"/>
    <param name="opacity"/>
</page>
<page name="Textures">
    <param name="ColMap" widget="filename"/>
<param name="SpecMap" widget="filename"/>
<param name="Repeats" widget="boolean"/>
<param name="RepeatT" widget="boolean"/>
</page>
<page name="Bump Mapping">
<param name="BumpMap" widget="filename"/>
<param name="BumpVal"/>
</page>
<page name="Reflection">
<param name="EnvMap" widget="filename"/>
<param name="EnvVal"/>
<param name="UseFresnel" widget="boolean"/>
</page>
    <page name="Refraction">
        <param name="RefractMap" widget="filename"/>
<param name="RefractVal"/>
<param name="RefractEta"/>
```

```
</page>
</args>
```

Args Files for Renderer Procedurals

Similar to the use in shader interfaces, UI hints can be defined for RenderMan procedurals. The `RendererProceduralArgs` node looks for an **.args** file called `<proceduralName>.so.args` in the same directory as the **.so** for the procedural.

This shows an example of a `procedural.so.args` file:

```
<args format="1.0" outputStyle="scenegraphAttr">
<float name='color' size='3' default='1,0,0'
widget='color'/>
<float name='radius' default='1'/>
</args>
```



NOTE: There are two output styles for how parameters to be parsed to procedurals are serialized into a string:

1. Serialized key-value pairs of the parameters. This is used by default when no `outputStyle` is specified.
2. XML serialization of the parameters as `ScenegraphAttributes` (as used in the example above).



NOTE: For procedurals, the type and default value of a parameter have to be declared. This is in contrast to the use of Args Files in Shaders where the type can be interrogated directly from the shader.

UI Hints for Plug-ins Using Argument Templates

Instead of using **.args** files, Scene Graph Generators (SGG) and Attribute Modifier Plugins (AMP) must declare UI hints directly in their source code as part of the Argument Template. The Argument Template is used to declare what arguments need to be passed to the plug-in.

The syntax used for these is the same as you would use in an **.args** file, just that you're handing the values as attributes instead of declaring them inside an **.xml** file.

Usage in Python Nodes

In Python, additional UI hints, such as widget or help text, can be specified by defining them in a dictionary. The dictionary is then passed to Katana in the **addParameterHints** function (part of the Nodegraph API).

Here is an example of how this is done for the `Alembic_In` node:

```
_ExtraHints = {
"Alembic_In.name" : {
```

```

"widget" : "newScenegraphLocation",
},
'Alembic_In.abcAsset': {
'widget': 'assetIdInput',
'assetTypeTags': 'geometry|alembic',
'fileTypes': 'abc',
'help': ""Specify the asset input for an Alembic
(.abc) file.""
},
}

```

Usage in C++ Nodes

In C++ nodes, the Argument Template consists of (nested) groups containing the UI hints. Each UI element has its group of hints, which is then added to a top-level group. The resulting hierarchy for a simple example using a checkbox, file chooser, and dropdown might look as follows:

```

+ top-level group
| + checkBoxArg (float)
| + checkBoxArg__hints (group)
| | + widget (string)
| | + help (string)
| | + page (string)
| + fileArg (string)
| + fileArg__hints (group)
| | + widget (string)
| | + help (string)
| | + page (string)
| + dropBoxArg (string)
| + dropBoxArg__hints (group)
| | + widget (string)
| | + options (string)
| | + help (string)
| | + page (string)

```

The following example code shows the implementation of the hierarchy shown above, and how the top-level group is built and returned in **getArgumentTemplate**:

```

static FnKat::GroupAttribute getArgumentTemplate()
{
    FnKat::GroupBuilder gb_checkBoxArg_hints;
    gb_checkBoxArg_hints.set("widget",
    FnKat::StringAttribute( "checkBox" ) );
    gb_checkBoxArg_hints.set("help",

```

```

FnKat::StringAttribute( "the mode value" ) );
gb_checkBoxArg_hints.set("page",
FnKat::StringAttribute( "pageA" ) );

FnKat::GroupBuilder gb_fileArg_hints;
gb_fileArg_hints.set("widget",
FnKat::StringAttribute( "assetIdInput" ) );
gb_fileArg_hints.set("help",
FnKat::StringAttribute( "the file to load" ) );
gb_fileArg_hints.set("page",
FnKat::StringAttribute( "pageA" ) );

FnKat::GroupBuilder gb_dropBoxArg_hints;
gb_dropBoxArg_hints.set("widget",
FnKat::StringAttribute( "mapper" ) );
gb_dropBoxArg_hints.set("options",
FnKat::StringAttribute( "No:1|SmoothStep:2|
InverseSquare:3" ) );
gb_dropBoxArg_hints.set("help",
FnKat::StringAttribute( "a dropbox argument" ) );
gb_dropBoxArg_hints.set("page",
FnKat::StringAttribute( "pageA" ) );

FnKat::GroupBuilder gb;
gb.set("checkBoxArg", FnKat::FloatAttribute( DEFAULT_SCALE ) );
gb.set("checkBoxArg__hints",
gb_checkBoxArg_hints.build() );
gb.set("fileArg", FnKat::StringAttribute(
"/tmp/myFile.xml" ) );
gb.set("fileArg__hints", gb_fileArg_hints.build() );
gb.set("dropBoxArg", FnKat::StringAttribute( "No" ) );
gb.set("dropBoxArg__hints", gb_dropBoxArg_hints.build() );
    return gb.build();
}

```


AttributeFiles

Overview

This is a simple format that uses XML files to describe additional attributes that need to be created at specified scenegraph locations. A Katana node called 'AttributeFile_In' is used to read in the XML file and create the new attributes.

The file specifies names of scenegraph locations where new attributes should be created, and the values of those attributes. The XML file specifies a number attributeLists. Each of these specify a location name, and a number of attribute-value pairs to be created at locations that match that name.

These attributes can be used for various purposes, including specifying texture names to be used on geometry. The filter can also use custom file parsers to read custom file formats or read the required attribute data from another source.

Current Limitations

- Only string attributes are supported.
- No animation for attributes.
- Location matching is only by simple explicit location names. No wildcards or more complex location name matching.
- At present, new file parsers have to be written by The Foundry. Users are able to create their own when the plug-in APIs are exposed.

Usage in Katana

The 'AttributeFile_In' node is used to read in the attribute file and create new attributes on the required scenegraph locations.

Parameters:

- **CEL** - specifies the locations to run this filter on. Typically this is the location of an asset, such as `/root/world/geo/myAsset//*`, which you want to dress the new attributes on.
- **File Path** - name of the attributes file to read in.
- **Custom File Parser** - allows specification of an `.so` file for a custom file parser other than the default one for XML AttributeFiles.
- **Attribute Group Name** - specifies the Name for the Group Attribute where the attributes from the file are stored. Leave it empty to store the attributes directly under the location (without a group attribute).

- **Apply When** - specifies when the filter is to be run (immediately or using a resolver). Uses similar options to the Katana AttributeScript node.

Example Scene Using AttributeFile

In `katana_alpha_demo` there is a Katana project file called `'houseScene_textured.katana'`. This shows an XML AttributeFile being used to create new attributes, and a Python process being executed using an AttributeScript to change those attributes into full texture file paths needed by PRMan.

This scene file is designed to run as it is if the `katana_alpha_demos` are placed into `/home/katana`. If you place the scenes into another directory you need to modify:

- the path used in the `ScenegraphXML_In` node to load the houseScene data.
- the path used in the `AttributeFile_In` node to load the XML AttributeFile.
- the path specified in the first line of the Python script in the `AttributeScript` node for the texture files.

Example of a Simple XML AttributeFile

```
<attributeFile version=" 0. 1. 0">
  <attributeList location="groundplaneShape">
    <attribute name="channel" type="string" value="ColorMap" />
    <attribute name="path" type="string" value="ground.tx" />
  </attributeList>

  <attributeList location=" leavesShape">
    <attribute name="channel" type="string" value="ColorMap" />
  <attribute name="path" type="string" value="leaves.tx" />
  </attributeList>

  <attributeList location="trunkShape">
    <attribute name="channel" type="string" value="ColorMap" />
  <attribute name="path" type="string" value="trunk.tx" />
  </attributeList>
</attributeFile>
```

ScenegraphXML

Overview

This is a simple format that uses a combination of XML, together with Alembic files, to represent structured hierarchical scenegraph data. It's designed to provide a format that can be used to read structured assets into Katana, and a reference solution for tools to bring other structured hierarchical data into Katana.

Two different types of scenegraph elements are used to create the scenegraph hierarchy - group instances and reference instances:

- A **group** has a number of child instances, all of which are declared within the same XML file. If the instance list is empty then a leaf location is created.
- A **reference** indicates that the child scenegraph is declared in a separate file. Currently, this can be another ScenegraphXML file or a geometry cache using Alembic.

Since one ScenegraphXML file can reference another, one feature of the system is that you can have multiple levels of referencing.

Each instance in the ScenegraphXML file can also have the following additional information:

- A 3D transform (Xform) represented using a 4x4 matrix (16 floats).
- Bounding box data (6 floats: minx, maxx, miny, maxy, minz, maxz).
- Level of detail data, used to control switching between different representations of an asset.
- Level of detail data can be either in the form of string 'tags' or floating point 'weight' values (or both).
- Paths to proxy geometry files, such as for use in Katana's OpenGL viewer.
- Arbitrary metadata using float, float lists, or strings.

Float and float list values in the ScenegraphXML files can be animated using simple single values per frame. These values are held in associated channel data XML files, with a separate file per frame. Any values that are animated are indicated in the ScenegraphXML file by a channel index value, giving the index within the file for the animated value, or the index of the first value for a list.

At the root of the ScenegraphXML is a list of the top level instances of the scenegraph hierarchies described by the file.



NOTE: Using ScenegraphXML all the geometry needs to be held in geometry caches. The XML files themselves can only hold hierarchical scenegraph structure, not actual geometry. In the current implementation the geometry caches have to be in Alembic format. See [Data Format Description](#) for more information on how the data is structured in the XML files.



NOTE: ScenegraphXML is provided as a reference example only. It is not intended for production use and is not supported.

Using ScenegraphXML in Katana

The ScenegraphXml_In node is used to bring structured hierarchical scenegraph data into Katana.

Parameters:

- **name** – the location in the Katana scene graph to bring the ScenegraphXML hierarchy in under.
- **filepath** – the filepath on disk for the top level ScenegraphXML file.

Reading and Writing ScenegraphXML from Python

The scenegraphXML.py file provides a set of Python classes to enable read and write ScenegraphXML format data. This Python script is provided as an extra utility for Katana, located in \${KATANA_ROOT}/extras/ScenegraphXML. In order to pick up this script this directory should be added to the PYTHONPATH before starting Maya.



NOTE: ScenegraphXML.py, maya2scenegraphXML.py, and other ScenegraphXML scripts are provided as reference examples only. They are not intended for production use and are not supported.

There are three main classes used to declare scenegraph data:

- **ScenegraphRoot** - This class holds a list of the top level instance nodes in the hierarchy, as well as having methods to read and write ScenegraphXML data, set animating channel values, and read and write per-frame channel files.
- **Group** - Class to represent group instances. Holds a list of child instances, that are other Group or Reference instances held locally. It can also contain data for bounding box information, level of detail data, proxies and arbitrary metadata (float, float list, or string).
- **Reference** - Class to represent reference instances. Similar to a group node except holds a reference to the file that needs to be read to get the child scenegraph instead of an instance list. Currently, two types of reference are supported:
 - 'xml' – another ScenegraphXML file.
 - 'abc' – a geometry cache hierarchy using Alembic format.

A single ScenegraphRoot instance needs to be created, and the hierarchy is build out of Group and Reference instances. Top level elements in the hierarchy are added to the instance list of the ScenegraphRoot.

Example of a simple Python script using ScenegraphXML.py:

```
import scenegraphXML as sgxml
```

```
# declare the ScenegraphRoot
root = sgxml.ScenegraphRoot()
```

```

# declare elements for the hierarchy
group1 = sgxml.Group( name="group1" )
group1.setBounds(value=[-1.0, 1.0, -1.0, 1.0, -1.0, 2.0])
group1.setXform(value=[1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1])

group2 = sgxml.Group( name="group2" )
group2.setXform(value=[1,0,0,0,0,1,0,0,0,0,1,0,1,2,3,1])

xmlref1 = sgxml.Reference(name='xmlref1',
refFile='/tmp/subscene1.xml' )
xmlref2 = sgxml.Reference(name='xmlref1',
refFile='/tmp/subscene2.xml' )
xmlref2.setXform(value=[1,0,0,0,0,1,0,0,0,0,1,0,10,20,30,1])

georef1 = sgxml.Reference(name='georef1', refType='abc',
refFile='pony.abc' )
georef1.setXform(value=[1,0,0,0,0,1,0,0,0,0,1,0,100,200,300,1])

georef2 = sgxml.Reference(name='georef2', refType='abc',
refFile='prim.abc' )

# connect up the hierarchy
root.setInstanceList([group1, group2])
group1.setInstanceList( [ xmlref1, xmlref2 ] )
group2.setInstanceList( [ georef1, georef2 ] )

# write out the hierarchy to disk
root.writeXMLFile('/tmp/scenegraph1.xml' )

```

See [ScenegraphXML.py Classes and Methods](#) for more information.

Writing ScenegraphXML data from Maya

The maya2scenegraphXML.py file provides some Python utilities to aid exporting from Maya 2010 and 2011 into ScenegraphXML format.

To use these tools you set various custom attributes on nodes in a Maya hierarchy to indicate things such as:

- Whether to export everything under a given node as a component using Alembic.
- Whether to split the hierarchy at a given node into a separate 'assembly' using a separate referenced ScenegraphXML file.

- Whether to set a proxy representation viewable when the scenegraph is exposed in Katana to a given location.
- Level of Detail data.
- Arbitrary metadata using floats or strings. A number of helper functions are provided to help set the required custom attribute values:

setComponent

Usage: `setComponent(selection, filepath=None, refType='abc')`

Sets the selected nodes in the Maya hierarchy as the top nodes of a 'component': a section of hierarchy stored as a geometry cache using Alembic. The selected nodes become references to Alembic files.

If a filepath to an existing **.abc** file is specified, that file is used for the referenced geometry.

If no filepath is specified, the `AbcExport` plug-in is called to automatically write out the children of the selected nodes as Alembic files (one file per selected Maya node). The names of the files are automatically generated based on the Maya paths of the nodes.

setStaticComponent

Usage: `setStaticComponent(selection, filepath=None, refType='abc')`

Functions in the same manner to `setComponent`, except only exports geometry caches for a single static frame instead of over an animation range.

setAssembly

Usage: `setAssembly(selection, filepath=None)`

Sets the selected node in the Maya hierarchy as the top nodes of an 'assembly': a section of hierarchy stored in a separate ScenegraphXML file. The selected nodes become references to Scenegraph XML files.

If a filepath to an **.xml** file is specified, that file is used for the referenced ScenegraphXML file.

If no filepath is specified, a separate ScenegraphXML file is written out for each sub-hierarchy. The name of the files is automatically generated based on the Maya paths of the nodes.

setReference

Usage: `setReference(selection, filepath, refType='xml')`

Sets the selected Maya nodes to be exported as reference instances to the specified file, and reference type `refType`. Valid values for `refType`: 'xml', 'abc' and 'tako'.

setLodGroup

Usage: setLodGroup(selection)

Sets the selected Maya nodes to be exported as Level of Detail groups. Conventionally, this is used as the parent of a number of Level of Detail nodes in Maya, each of which is assigned data used for Level of Detail switching.

setLodNode

Usage: setLodNode(selection, lodTag=None, lodWeight=None)

Sets the selected nodes to be exported with level of detail data, used in Katana to switch between level of detail representations.

Two types of level of detail data can be used:

- lodTag – string tags. Lod selection in Katana is by explicit use of the matching tag name.
- lodWeight – floating point values. Lod selection in Katana is by use of a weight value. The LoD with the closest value is selected.

setIgnore

Usage: setIgnore(selection)

Sets the selected Maya nodes (or any children of those nodes) to not export to ScenegraphXML.

setBoundsWriteMode

Usage: setBoundsWriteMode(selection, value=True)

Sets the selected Maya nodes to either be exported with or without bounds information. By default all nodes exported using maya2ScenegraphXML have bounds information.

enableCustomBounds

Usage: enableCustomBounds(selection, value=True)

Sets the selected nodes to be exported with bounds information values you supplied. The bounds values should be set using Maya attributes called:

- sgxml_boundMinX,
- sgxml_boundMaxX,
- sgxml_boundMinY,

- sgxml_boundMaxY,
- sgxml_boundMinZ, and
- sgxml_boundMaxZ.

setProxy

Usage: setProxy(selection, proxyFile, proxyName='viewer')

Sets a file to be used a proxy representation for the selected Maya nodes. The proxy file needs to be an Alembic geometry file.

The proxyName specifies the attribute name for the proxy in Katana. The default value of 'viewer' indicates that this is a proxy for use in Katana's OpenGL viewer.

setArbAttr

Usage: setArbAttr(selection, attrName, value, attrType='float')

Sets arbitrary attribute metadata to be exported on the selected Maya nodes. When imported into Katana, these attributes are named scenegraphXmlAttrs.<attrName> Currently only float and string types are supported for export from Maya. Float data can be animated.

deleteSgxmlAttrs

Usage: deleteSgxmlAttrs(mayaPath)

Deletes all the additional attributes used by ScenegraphXML from the Maya hierarchy under mayaPath.

maya2ScenegraphXML

Usage: maya2ScenegraphXML(mayaSelection, xmlFileName, startFrame=None, endFrame=None, arbAttrs=None, geoFileOptions="")

The main function that needs to be called to actually export from Maya into Scenegraph XML. Exports the hierarchies under the selected nodes, using xmlFileName for the top level ScenegraphXML file.

arbAttrs: list of strings that allow specifies the names of any arbitrary attributes that should be exported into the ScenegraphXML files

geoFileOptions: a string that allows additional options to be passed to AbcExport, such as to export additional attributes to the Alembic file.

Alembic Maya Plug-Ins

To assist exporting Alembic files we have provided compiled plug-ins for Maya 2010 and Maya 2011.

These plug-ins are provided in the 'extras' directory in the Katana installation.

Example Maya Export Scripts

Example Python scripts to export from Maya using `maya2scenegraphXML.py`, together with accompanying Maya scenes, are also provided in `demos/maya_files`

`houseExport.py`

Shows how to export a simple Maya hierarchy and geometry. The scene can be found in `demos/maya_files/scenes/houseScene.ma`

In this scene there are 4 elements: house, tree, ground and smoke. Every one of these elements is a component with the exception of the house, which is an assembly that contains 2 other components: `houseBottom` and `houseTop`. Some components, the tree, for example, contain a sub-hierarchy with more than one object, such as leaves and trunk, in the case of the tree.

To export the scene from Maya the `demos/maya_files/scripts/houseSceneExport.py` should be imported into Maya's script editor with the scene already loaded - `import houseSceneExport`.

The output is written in `/tmp/houseScene/` where the main XML file can be found: `houseScene.xml`.

`robotExport_abc.py`

A more complex example might be where a rigged animated object is exported from Maya, including a number of features, such as proxy Alembic files, low and high levels of detail, multiple 'assembly' levels of referenced ScenegraphXML files, and export of arbitrary attributes (both to ScenegraphXML and in the Alembic files used for components).

The `robotExport_abc.py` script should be found in `demos/maya_files/scripts`. To run the example, simply load the Maya scene and run the Python `robotExport_abc.py` script from within Maya.

By default the script writes the output files into `/tmp/robot_data_abc`.

The proxies set in the script assumes that **.abc** files for the proxies are available the export directory. If you want to re-use the proxy files from the supplied demo assets, simply copy all the `*proxy.abc` files from `demos/robot_data_abc` to the export directory.

ScenegraphXML attributes in Maya

The additional attributes that maya2scenegraphXML uses to control how the nodes in the hierarchy are exported have the prefix 'sgxml_'.

Arbitrary attributes for export into ScenegraphXML files have the prefix 'arbAttr_'.

Custom Attributes in Alembic Components

To export arbitrary attributes in Alembic files used for the components, simply create the attributes on group or shape nodes in Maya and specify the names of the attributes you want to export as part of the geoFileOptions when calling maya2scenegraphXML.

For example to export an attribute called 'ColMapTag' in an Alembic file, include the option '-attr ColMapTag'.

Alembic files can support arbitrary attributes of several types, including floating point numbers, integers, strings, booleans, and lists of various types.

Data Format Description

Notation:

* - optional

... - one or more

...* - zero or more

ScenegraphXML

Root

```
<scenegraphXml name="building"* version="0.1.0"*>
<channelData startFrame="1" endFrame="10"
ref="/tmp/myChannelData" */>*
<instanceList/>
</xmlScene>
```

InstanceList

```
<instanceList>
<instance/>*...
```

```
</instanceList>
```

Instances

base types: group, reference

```
<instance type="group" name="blah">
<instanceList/>*
<bounds/>*
<xform/>*
<proxyList/>*
<lookFile/>*
<modifiers/>*
<arbitraryList/>*
<lodData tag="hi" weight="0.75"/>*
</instance>
```

```
<instance type="group" groupType="lodGroup" name="blah">
<instanceList/>
<bounds/>*
<xform/>*
<proxyList/>*
<lookFile/>*
<modifiers/>*
<arbitraryList/>*
<lodData tag="hi" weight="0.75"/>*
</instance>
```

```
<instance type="reference" name="blah"
refFile="subScene.xml">
<bounds/>*
<xform/>*
<proxyList/>*
<lookFile/>*
<modifiers/>*
<arbitraryList/>*
<lodData tag="hi" weight="0.75"/>*
</instance>
```

```
<instance type="reference" name="blah" refType="abc"
refFile="myGeometry. abc">
<bounds/>*
```

```

<xform/>*
<proxyList/>*
<lookFile/>*
<modifiers/>*
<arbitraryList/>*
<lodData tag="hi" weight="0.75"/>*
</instance>

```



NOTE: If no refType is explicitly specified in for a reference instance, it's assumed that the reference is to another XML file containing ScenegraphXML data.

Xform

```
<xform value="1 0 0 0 0 1 0 0 0 0 1 0 10 20 30 1"/>
```

or

```
<xform channelIndex="6"/>
```

Bounding Boxes

```

<bounds minx="-10" maxx="10" miny="-10" maxy="10"
minz="-10" maxz="10"/>

```

or

```
<bounds channelIndex="6"/>
```

ProxyList

```

<proxyList>
<proxy name="viewer" ref="/tmp/proxyFile.abc"/>...
</proxyList>

```

ArbitraryList (metadata)

```

<arbitraryList>
<attribute name="myAttr" type="float" value="10.0"/>...
<attribute name="myAttr" type="float"
channelIndex="6"/>...
<attribute name="myAttr" type="floatlist"
value="1.0 2.0 3.0 4.0"/>...
<attribute name="myAttr" type="floatlist"
numValues="6" channelIndex="10"/>...
<attribute name="myAttr" type="string" value="Pony"/>...
</arbitraryList>

```

attribute types: float, floatlist, string

Look File

```
<lookFile ref="myLookFile.klf"/>
```



NOTE: LookFile assignment through ScenegraphXML files currently only works for files imported using the Importomatic. LookFiles can only be associated with locations that are references to other SgXML files or explicitly set the groupType to 'assembly'.

Modifiers

```
<modifiers>
<attributeFile/>...*
</modifiers>
```

Attribute File

Default XML parser: `<attributeFile ref="myAttributeFile.xml"/>`

Custom XML parser: `<attributeFile ref="myAttributeFile.xml" customParser="myParser.so"/>`

The metadata is stored under the group name **attributeFile** by default. It is possible to override this by using the parameter **groupName**. For instance: `<attributeFile ref="myAttributeFile.xml" groupName="myGroupName"/>`

Channel Data Files

A simple format to represent animation of float values in using **.xml** files with multiple channels held in each file, and a single **.xml** file per frame.

```
<channels>
<c v="0.1"/>...
</channels>
```

ScenegraphXML.py Classes and Methods

Class: ScenegraphRoot

```
ScenegraphRoot( name=None, instanceList=None, xform=None,
bounds=None, proxyList=None,
arbitraryList=None, channelData=None)
```

Methods:

```

ScenegraphRoot.setInstanceList(instanceList)
ScenegraphRoot.addInstance(newInstance)
ScenegraphRoot.addInstances(newInstances)
ScenegraphRoot.setChannelDataValue(index, value)
ScenegraphRoot.calcBounds()
ScenegraphRoot.writeXMLFile(filepath, verbose=True)
ScenegraphRoot.readXMLFile(filepath)
ScenegraphRoot.writeXMLChannelFile(frameNumber)
ScenegraphRoot.readXMLChannelFile(frameNumber)

```

Class: Group

```

Group( name=None, instanceList=None, groupType=None,
xform=None, bounds=None, proxyList=None,
arbitraryList=None)

```

Methods:

```

Group.setInstanceList(instanceList)
Group.addInstance(newInstance)
Group.addInstances(newInstances)
Group.setBounds(minx=None, maxx=None, miny=None, maxy=None,
minz=None, maxz=None, value=None,
channelIndex=None)
Group.getBounds(channelData)
Group.setXform(value=None, channelIndex=None)
Group.getXform(channelData)
Group.setLodData(tag=None, weight=None, channelIndex=None):
Group.addProxy(name, ref)
Group.addArbitraryAttribute(name, dataType, value=None,
numValues=None, channelIndex=None)

```

Class: Reference

```

Reference(name=None, refType='xml', refFile=None,
bounds=None, proxyList=None, arbitraryList=None)

```

Methods:

```

Reference.setReference(refType, refFile)
Reference.setRefFile(refFile)
Reference.setRefType(refType)
Group.setBounds(minx=None, maxx=None, miny=None, maxy=None,
minz=None, maxz=None, value=None,
channelIndex=None)

```

```
Group.getBounds(channelData)
Group.setXform(value=None, channelIndex=None)
Group.getXform(channelData)
Group.setLodData(tag=None, weight=None, channelIndex=None)
Group.addProxy(name, ref)
Group.addArbitraryAttribute(name, dataType, value=None,
numValues=None,
channelIndex=None)
```

Appendix G: End User License Agreement

End User License Agreement (EULA)

PLEASE READ THIS EULA CAREFULLY BEFORE ORDERING OR DOWNLOADING OR USING ANY FOUNDRY SOFTWARE. YOUR ATTENTION IS PARTICULARLY DRAWN TO CLAUSES 12 AND 13 WHERE WE LIMIT OUR LIABILITY TO USERS OF OUR SOFTWARE PRODUCTS.

IMPORTANT NOTICE TO ALL USERS: BY DOWNLOADING AND/OR USING THIS SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THIS EULA, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THE TERMS OF THIS EULA DO NOT DOWNLOAD, INSTALL, COPY OR USE THE SOFTWARE.

IMPORTANT NOTICE TO CONSUMERS WHO PURCHASE SOFTWARE DIRECT FROM THE FOUNDRY: YOU HAVE THE RIGHT TO WITHDRAW FROM YOUR TRANSACTION WITH THE FOUNDRY WITHOUT CHARGE AND WITHOUT REASON AT ANY TIME BEFORE DOWNLOADING OUR PRODUCT(S). HOWEVER YOU WILL LOSE THIS RIGHT ONCE YOU BEGIN TO DOWNLOAD OUR PRODUCT(S). THIS DOES NOT AFFECT YOUR CONSUMER RIGHTS IN RELATION TO DEFECTIVE PRODUCTS OR SERVICES.

This END USER LICENSE AGREEMENT (this "EULA") is, in cases where you purchase our product(s) direct from The Foundry, incorporated into the agreement between The Foundry Visionmongers Limited, a company registered in England and Wales, ("The Foundry"), and you, as either an individual or a single company or other legal entity ("Licensee") on the terms of which you will purchase the products and services of The Foundry (the "Agreement"). In cases where you purchase our product(s) from one of our resellers, the use of the term "Agreement" in this EULA refers to the arrangements between The Foundry and Licensee on which Licensee is permitted to use The Foundry's product(s), including this EULA.

The Foundry reserves the right to refuse to grant a License (as defined in clause 1.1) to any Licensee who has failed to pay any sum due either to The Foundry or to a reseller of The Foundry either in connection with the Agreement or in connection with any other software license to use any product(s) of The Foundry.

1. GRANT OF LICENSE

1.1 Subject to the limitations of clause 3 and all the other terms of the Agreement, The Foundry grants to Licensee a limited, non-transferable (subject to clause 2.1 (b) below) and non-exclusive license to download, install and use a machine readable, object code version (subject to clauses 3 and 4 below) of the software program(s) purchased by

Licensee (the "Software") and any accompanying user guide and other documentation (the "Documentation"), solely for Licensee's own internal purposes (the "License"); provided, however, that Licensee's right to download, install and use the Software and the Documentation is limited to those rights expressly set out in this EULA.

1.2 Some types of license models set out in clause 2.1 limit the installation and use of the Software to the country in which Licensee is based at the date of purchase (the "Home Country"), unless otherwise agreed in writing.

Notwithstanding such limits, Licensee may still use the Software outside the Home Country if traveling or working outside the Home Country on a temporary basis provided that such use does not exceed 70 days in aggregate in any rolling twelve month period or, in the case of any license which lasts for less than twelve months, does not exceed the number of days representing 20% of the term of the license.

1.3 Only to the extent that is proportionate to, and reasonably necessary to support, Licensee's licensed use of the Software in accordance with the Agreement, Licensee may (provided valid license keys have been obtained) install the Software on more than one computer, provided always that Licensee's concurrent use of different installations of the Software does not exceed the number of valid Licenses that Licensee has paid for or licensed (as applicable).

2. LICENSE MODELS

2.1 For each product purchased from The Foundry, the License will be one of the following types of license, and subject to the following terms and conditions. Please note that some licensing models set out below do not apply to certain products of the Foundry. Whichever licensing model applies, Licensee shall not at any one time use more copies of the Software than the total number of valid licenses purchased or licensed by Licensee (as applicable).

(a) "Node Locked License"

If Licensee purchases a Node Locked License, Licensee will install and use only a single copy of the Software on only one computer at a time, which may be located anywhere in the Home Country.

(b) "Individual License"

If Licensee purchases an Individual License, Licensee warrants and represents that Licensee is a natural person and that only Licensee will use the Software. Licensee may transfer or assign ("transfer") the Individual License to another natural person ("Assignee") subject to Licensee: (i) notifying The Foundry of such transfer and obtaining The Foundry's express written consent, (ii) paying an administrative fee with respect to such transfer as may be required by The Foundry, and (iii) after transferring a single copy of the Software to the Assignee, deleting any copies of the Software that Licensee may have in Licensee's possession, custody or power. An Individual License entitles Licensee to use the Software on only one computer at a time, which may be located anywhere and is not restricted to the Home Country.

(c) "Floating License"

If Licensee purchases a Floating License, use of the Software may be at any site in the Home Country.

(d) "Login-Based License"

If Licensee purchases a Login-Based License, Licensee warrants and represents that Licensee is a natural person and that only Licensee shall use the Software. Licensee will be issued with log in details and may use the Software on any number of computers (but not simultaneously).

2.2 Some of the Software may be made available at concessionary rates or free of charge (as applicable) as follows:

(a) “Educational License”

If Licensee has purchased the Software on the discounted terms of The Foundry’s Educational Policy published on its website (the “Educational Policy”), Licensee warrants and represents to The Foundry as a condition of the Educational License that: (i) (if Licensee is a natural person) he or she is a part-time or full-time student at the time of purchase and will not use the Software for any commercial, professional or for-profit purposes; (ii) (if the Licensee is not a natural person) it is an organization that will use the Software only for the purpose of training and instruction, and for no other purpose, and (iii) Licensee will at all times comply with the Educational Policy (as such policy may be amended from time to time). Unless the Educational License is a Floating License, Licensee shall use the Software on only one computer at a time.

(b) “Non-Commercial License”

If the License is a Non-Commercial License, Licensee warrants and represents that Licensee is a natural person, that they will only access and/or use one copy of a Non-Commercial License for personal, recreational and non-commercial purposes and that only Licensee will use the Software. Under a Non-Commercial License, Licensee will not use the Software: (a) in conjunction with any other copies or versions of the Software, under any type of License model; (b) for any commercial, professional, for-profit and/or on-sale purpose or otherwise to provide any commercial service(s) to a third party (whether or not for financial or other reward and including for education, instruction of or demonstration to any third party for commercial purposes); (c) in the course of any employment or business undertaking of Licensee; (d) on any commercial premises during business hours (except where use of the Software is solely for a personal, recreational, educational or other non-commercial purpose); and/or (e) to create any commercial tools or plug ins.

(c) “MODO Steam Edition”

A version of MODO with limited functionality as described in the Documentation is available to purchase on discount terms through Valve Corporation’s Steam store. If Licensee has purchased such version, Licensee warrants and represents to The Foundry as a condition of the Agreement that: (i) Licensee is a natural person; and (ii) Licensee will use the Software strictly through Steam and only for personal, recreational and non-commercial use, except only that if Licensee uses the Software to create assets and content Licensee may sell such assets and content through Valve’s Steam Workshop.

(d) “MODO indie” and “MARI indie”

Variants of MODO and MARI with limited functionality as described in the Documentation are available to purchase on discount terms through Valve Corporation’s Steam store. If Licensee has purchased such a variant, Licensee warrants and represents to The Foundry as a condition of the Agreement that: (i) Licensee is a natural person; or (ii) Licensee is an entity in the direct ownership of a single natural person; (iii) Licensee will only access and/or use one copy of either variant; and (iv) only Licensee will use the Software.

(e) "Trial License"

Licensee may register for a "Trial License" of the Software (not available for all products or in all regions or markets). A Trial License lasts a limited specified period on the expiry of which the Software will automatically cease to function. Licensee will use the Software on only one computer at a time.

(f) "Free License"

Licensee may register for a "Free License" of selected Software from The Foundry (not available for all products or in all regions or markets). A Free License lasts for a limited specified period on the expiry of which the Software will cease to function. Usually, a replacement License to cover a new, time limited, period will be issued by the Foundry. Licensee will use the Software under a Free License on only one computer at a time.

(g) "Personal Learning Edition License"

If the Software is a Personal Learning Edition ("PLE"), it will not require a license key to be issued to Licensee and will have limited functionality as described in the Documentation. Licensee may use it only for the purpose of personal or internal training and instruction, and for no other purpose. PLE versions of the Software may not be used for commercial, professional or for-profit purposes including, for the avoidance of doubt, the purpose of providing training or instruction to third parties. Licensee shall use the Software on only one computer at a time.

2.3 If Licensee has purchased a License that permits "non-interactive" use of the Software ("Headless Rendering"), Licensee is authorized to use a non-interactive version of the Software for rendering purposes only (i.e. without a user, in a non-interactive capacity) and shall not use such Software on workstations or otherwise in a user-interactive capacity. Headless Rendering is not available on all products. In all cases except MODO (in respect of which there is no limit on the amount of Headless Rendering allowed), Headless Rendering licenses are limited to one computer such that the number of computers on which Headless Rendering can be carried out is limited to the number of valid Licenses that have been purchased.

3. RESTRICTIONS ON USE

Please note that in order to guard against unlicensed use of the Software a license key is required to access and enable the Software (other than Software which is licensed under the Personal Learning Edition model – see clause 2.2 (b) above). Licensee is authorized to use the Software in machine readable, object code form only (subject to clause 4), and Licensee shall not: (a) assign, sublicense, sell, distribute, transfer, pledge, lease, rent, lend, share or export the Software, the Documentation or Licensee's rights under this EULA; (b) alter or circumvent the license keys or other copy protection mechanisms in the Software or reverse engineer, decompile, disassemble or otherwise attempt to discover the source code of the Software; (c) (subject to clause 4) modify, adapt, translate or create derivative works based on the Software or Documentation; (d) use, or allow the use of, the Software or Documentation on any project other than a project produced by Licensee (an "Authorized Project") or to provide a service (whether or not any charge is made) to any third party; (e) allow or permit anyone (other than Licensee and Licensee's authorized employees to the extent they are working on an Authorized Project) to use or have access to the Software or Documentation; (f) copy or install the Software or Documentation other than as expressly provided for in this EULA; or (g) take any action, or fail to take action, that could adversely affect the trademarks, service marks, patents, trade secrets, copyrights or other intellectual property rights of The Foundry or any third party with

intellectual property rights in the Software (each, a "Third Party Licensor"). For purposes of this clause 3, the term "Software" shall include any derivatives of the Software.

Unless Licensee has purchased an Individual License or a Login-Based License, if the Software is moved from one computer to another, the issuing of replacement or substituted license keys is subject to and strictly in accordance with The Foundry's License Transfer Policy, which is available on The Foundry's website and which requires a fee to be paid in certain circumstances. The Foundry may from time to time and at its sole discretion vary the terms and conditions of the License Transfer Policy.

4. SOURCE CODE

Notwithstanding that clause 1 defines "Software" as an object code version and that clause 3 provides that Licensee may use the Software in object code form only:

4.1 if The Foundry has agreed to license to Licensee (including by way of providing SDKs, upgrades, updates or enhancements/customization) source code or elements of the source code of the Software, the intellectual property rights in which belong either to The Foundry or to a Third Party Licensor ("Source Code"), Licensee shall be licensed to use the Source Code as Software on the terms of this EULA and: (a) notwithstanding clause 3 (c) Licensee may use the Source Code at its own risk in any reasonable way for the limited purpose of enhancing its use of the Software solely for its own internal business purposes and in all respects in accordance with this EULA; (b) Licensee shall in respect of the Source Code comply strictly with all other restrictions applying to its use of the Software under this EULA as well as any other restriction or instruction that is communicated to it by The Foundry at any time during the Agreement (whether imposed or requested by The Foundry or by any Third Party Licensor);

4.2 to the extent that the Software links to any open source software libraries ("OSS Libraries") that are provided to Licensee with the Software, nothing in the Agreement shall affect Licensee's rights under the licenses on which the relevant Third Party Licensor has licensed the OSS Libraries, as stated in the Documentation. To the extent that Third Party Licensors have licensed OSS Libraries on the terms of v2.1 of the Lesser General Public License issued by the Free Software Foundation (see <http://www.gnu.org/licenses/lgpl-2.1.html>) (the "LGPL"), those OSS Libraries are licensed to Licensee on the terms of the LGPL and are referred to in this clause 4.2 as the LGPL Libraries. The Foundry will at any time during the three year period starting on the date of the Agreement, at the request of Licensee and subject to Licensee paying to The Foundry a charge that does not exceed The Foundry's costs of doing so, provide Licensee with the source code of the LGPL Libraries (the "LGPL Source") in order that Licensee may modify the LGPL Libraries in accordance with the LGPL, together with certain object code of the Software necessary to enable Licensee to re-link any modified LGPL Library to the Software (the "Object"); and

4.3 notwithstanding any other term of the Agreement The Foundry gives no express or implied warranty, undertaking or indemnity whatsoever in respect of the Source Code, the OSS Libraries (including the LGPL Libraries), the LGPL Source or the Object, all of which are licensed on an "as is" basis, or in respect of any modification of the Source Code, the OSS Libraries (including the LGPL Libraries) or the LGPL Source made by Licensee ("Modification"). Licensee may not use the Object for any purpose other than its use of the Software in accordance with this EULA. Notwithstanding any other term of the Agreement, The Foundry shall have no obligation to provide support, maintenance, upgrades or updates of or in respect of any of the Source Code, the OSS Libraries (including the LGPL Libraries), the LGPL Source, the Object or any Modification. Licensee shall indemnify The Foundry against all liabilities

and expenses (including reasonable legal costs) incurred by The Foundry in relation to any claim asserting that any Modification infringes the intellectual property rights of any third party.

5. BACK-UP COPY

Licensee may store one copy of the Software and Documentation off-line and off-site in a secured location within the Home Country that is owned or leased by Licensee in order to provide a back-up in the event of destruction by fire, flood, acts of war, acts of nature, vandalism or other incident. In no event may Licensee use the back-up copy of the Software or Documentation to circumvent the usage or other limitations set forth in this EULA.

6. OWNERSHIP

Licensee acknowledges that the Software (including, for the avoidance of doubt, any Source Code that is licensed to Licensee) and Documentation and all related intellectual property rights and other proprietary rights are and shall remain the sole property of The Foundry and the Third Party Licensors. Licensee shall not remove, or allow the removal of, any copyright or other proprietary rights notice included in and on the Software or Documentation or take any other action that could adversely affect the property rights of The Foundry or any Third Party Licensor. To the extent that Licensee is authorized to make copies of the Software or Documentation under this EULA, Licensee shall reproduce in and on all such copies any copyright and/or other proprietary rights notices provided in and on the materials supplied by The Foundry hereunder. Nothing in the Agreement shall be deemed to give Licensee any rights in the trademarks, service marks, patents, trade secrets, confidential information, copyrights or other intellectual property rights of The Foundry or any Third Party Licensor, and Licensee shall be strictly prohibited from using the name, trademarks or service marks of The Foundry or any Third Party Licensor in Licensee's promotion or publicity without The Foundry's express written approval.

Subject to clause 4.3, The Foundry undertakes (the "Undertaking") to defend Licensee or at The Foundry's option settle any claim brought against Licensee alleging that Licensee's possession or use of the Software or Documentation in accordance with the Agreement infringes the intellectual property rights of a third party in the same country as Licensee ("Claim") and shall reimburse all reasonable losses, damages, costs (including reasonable legal fees) and expenses incurred by or awarded against Licensee in connection with any such Claim, provided that the undertaking shall not apply where the Claim in question is attributable to possession or use of the Software or Documentation other than in accordance with the Agreement, or in combination with any hardware, software or service not supplied or specified by The Foundry. The Undertaking is conditional on Licensee giving written notice of the Claim to The Foundry as soon as reasonably possible, cooperating in the defence of the Claim and not making any admission of liability or taking any step prejudicial to the defence of the Claim. If any Claim is made, or in The Foundry's reasonable opinion is likely to be made, against Licensee, The Foundry may at its sole option and expense (a) procure for Licensee the right to continue using the Software, (b) modify the Software so that it ceases to be infringing, (c) replace the Software with non-infringing software, or (d) terminate the Agreement immediately by notice in writing to Licensee and refund the License Fee (less a reasonable sum in respect of Licensee's use of the Software to the date of termination) on return of the Software and all copies. The Undertaking constitutes Licensee's exclusive remedy and The Foundry's only liability in respect of Claims.

7. LICENSE FEE

7.1 Licensee acknowledges that (subject to clause 7.2) the rights granted to Licensee under this EULA are conditional on Licensee's payment in full of the license fee payable in connection with the Agreement or, as the case may be, payable to The Foundry's reseller (the "License Fee").

7.2 In the cases of Non-Commercial NUKE, Trial Licenses and Free Licenses, for the avoidance of doubt, the fact that no License Fee may be payable shall not be construed as a waiver by The Foundry of any right or remedy available to it in relation to any breach by Licensee of this EULA or the Agreement, or of any other right or remedy arising under applicable law, all of which are expressly reserved.

8. UPGRADES/ENHANCEMENTS

If the Licensee has paid an annually renewable fee for access to support, upgrades and updates for the Software ("Annual Upgrade and Support Programme"), this is subject to the terms and conditions for the Annual Upgrade and Support Programme available on The Foundry's website. The Foundry may from time to time and at its sole discretion vary the terms and conditions of the Annual Upgrade and Support Programme. The Annual Upgrade and Support Programme is not available for all license types and variations.

9. TAXES AND DUTIES

Licensee agrees to pay, and indemnify The Foundry from claims for, any local, state or national tax (exclusive of taxes based on net income), duty, tariff or other impost related to or arising from the transaction contemplated by the Agreement.

10. LIMITED WARRANTY

10.1 Subject to clause 10.3, The Foundry warrants that, for a period of ninety (90) days after Licensee first downloads the Software ("Warranty Period"): (a) that the Software will, when properly used on an operating system for which it was designed, perform substantially in accordance with the functions described in the Documentation; and (b) that the Documentation correctly describe the operation of the Software in all material respects. If, within the Warranty Period, Licensee notifies The Foundry in writing of any defect or fault in the Software as a result of which it fails to perform substantially in accordance with the Documentation, The Foundry will, at its sole option, either repair or replace the Software, provided that Licensee makes available all the information that may be necessary to identify, recreate and remedy the defect or fault. This warranty will not apply to any defect or fault caused by unauthorised use of or any amendment made to the Software by any person other than The Foundry. If Licensee is a consumer, the warranty given in this clause is in addition to Licensee's legal rights in relation to any Software or Documentation that is faulty or not as described.

10.2 The Foundry does not warrant that the Software or Documentation will meet Licensee's requirements or that Licensee's use of the Software will be uninterrupted or error free.

10.3 If Licensee purchases a license of the Software that is of a fixed term duration, the Warranty Period in clause 10.1 shall apply only to Licensee's first purchase of such license and not to any subsequent renewal(s) even if a renewal involves another download.

11. INDEMNIFICATION

Licensee agrees to indemnify, hold harmless and defend The Foundry, the Third Party Licensors and The Foundry's and each Third Party Licensor's respective affiliates, officers, directors, shareholders, employees, authorized resellers, agents and other representatives from all claims, defence costs (including, but not limited to, legal fees), judgments, settlements and other expenses arising from or connected with any claim that any authorised or unauthorised modification of the Software or Documentation by Licensee or any person connected with Licensee infringes the intellectual property rights or other proprietary rights of any third party.

12. LIMITATION OF LIABILITY TO BUSINESS USERS

This clause applies where Licensee is a business user. Licensee acknowledges that the Software has not been developed to meet its individual requirements, and that it is therefore Licensee's responsibility to ensure that the facilities and functions of the Software as described in the Documentation meet such requirements. The Software and Documentation is supplied only for Licensee's internal use for its business, and not for any re-sale purposes or for the provision of services to third parties. The Foundry shall not under any circumstances whatever be liable to Licensee, whether in contract, tort (including negligence), breach of statutory duty, or otherwise, arising under or in connection with the Agreement for loss of profits, sales, business, or revenue, business interruption, loss of anticipated savings, loss or corruption of data or information, loss of business opportunity, goodwill or reputation or any indirect or consequential loss or damage. In respect of any other losses, The Foundry's maximum aggregate liability under or in connection with the Agreement whether in contract, tort (including negligence) or otherwise, shall in all circumstances be limited to the greater of US\$5000 and a sum equal to the License Fee. Nothing in the Agreement shall limit or exclude our liability for death or personal injury resulting from our negligence, fraud or fraudulent misrepresentation or for any other liability that cannot be excluded or limited by applicable law. This EULA sets out the full extent of our obligations and liabilities in respect of the supply of the Software and Documents. Except as expressly stated in this EULA, there are no conditions, warranties, representations or other terms, express or implied, that are binding on The Foundry. Any condition, warranty, representation or other term concerning the supply of the Software and Documentation which might otherwise be implied into, or incorporated in, the Agreement, whether by statute, common law or otherwise, is excluded to the fullest extent permitted by law.

13. LIMITATION OF LIABILITY TO CONSUMERS

This clause applies where Licensee is a consumer. Licensee acknowledges that the Software has not been developed to meet Licensee's individual requirements, and that it is therefore Licensee's responsibility to ensure that the facilities and functions of the Software as described in the Documentation meet such requirements. The Software and Documentation are only supplied for Licensee's domestic and private use. Licensee agrees not to use the Software and Documentation for any commercial, business or re-sale purposes, and The Foundry has no liability to Licensee for any loss of profit, loss of business, business interruption, or loss of business opportunity. The Foundry is only responsible for loss or damage suffered by Licensee that is a foreseeable result of The Foundry's breach of the

Agreement or its negligence but The Foundry is not responsible for any loss or damage that is not foreseeable. Loss or damage is foreseeable if they were an obvious consequence of a breach or if they were contemplated by Licensee and The Foundry at the time of forming the Agreement. Our maximum aggregate liability under or in connection with the Agreement, whether in contract, tort (including negligence) or otherwise, shall in all circumstances be limited to a sum equal to the greater of US\$5000 and a sum equal to the License Fee. Nothing in the Agreement shall limit or exclude our liability for death or personal injury resulting from our negligence, fraud or fraudulent misrepresentation or for any other liability that cannot be excluded or limited by applicable law.

14. TERM; TERMINATION

14.1 The Agreement is effective upon Licensee's download of the Software, and the Agreement will remain in effect until termination. Licensee may terminate the Agreement at any time on written notice to The Foundry. If Licensee breaches the Agreement, The Foundry may terminate the License by notice to Licensee. If the Agreement is terminated, the License will cease immediately and Licensee will either return to The Foundry all copies of the Software and Documentation in Licensee's possession, custody or power or, if The Foundry directs in writing, destroy all such copies. In the latter case, if requested by The Foundry, Licensee shall provide The Foundry with a certificate confirming that such destruction has been completed.

14.2 The Foundry reserves the right to terminate the License by notice to Licensee if it becomes aware that Licensee has failed to pay any sum due either to The Foundry or to a reseller of The Foundry either in connection with the Agreement or in connection with any other software license to use any product(s) of The Foundry.

15. CONFIDENTIALITY

Licensee agrees that the Software (including, for the avoidance of doubt, any Source Code that is licensed to Licensee) and Documentation are proprietary to and the confidential information of The Foundry or, as the case may be, the Third Party Licensors, and that all such information and any related communications (collectively, "Confidential Information") are confidential and a fundamental and important trade secret of The Foundry and/or the Third Party Licensors. If Licensee is a business user, Licensee shall disclose Confidential Information only to Licensee's employees who are working on an Authorized Project and have a "need-to-know" such Confidential Information, and shall advise any recipients of Confidential Information that it is to be used only as expressly authorized in the Agreement. Licensee shall not disclose Confidential Information or otherwise make any Confidential Information available to any other of Licensee's employees or to any third parties without the express written consent of The Foundry. Licensee agrees to segregate, to the extent it can be reasonably done, the Confidential Information from the confidential information and materials of others in order to prevent commingling. Licensee shall take reasonable security measures, which measures shall be at least as great as the measures Licensee uses to keep Licensee's own confidential information secure (but in any case using no less than a reasonable degree of care), to hold the Software, Documentation and any other Confidential Information in strict confidence and safe custody. The Foundry may request, in which case Licensee agrees to comply with, certain reasonable security measures as part of the use of the Software and Documentation. This clause shall not apply to any information that is in or comes into the public domain, or was in Licensee's lawful possession before receipt or which Licensee develops independently and without breach of this clause. Licensee acknowledges that monetary damages may not be a sufficient remedy for unauthorized disclosure of Confidential Information, and that The Foundry shall be

entitled, without waiving any other rights or remedies, to such injunctive or other equitable relief as may be deemed proper by a court of competent jurisdiction.

16. INSPECTION AND INFORMATION

16.1 Unless Licensee is a consumer, Licensee shall advise The Foundry on demand of all locations where the Software or Documentation is used or stored. Licensee shall permit The Foundry or its authorized agents to audit all such locations during normal business hours and on reasonable advance notice.

16.2 The Software may include mechanisms to collect limited information from Licensee's computer(s) and transmit it to The Foundry. Such information (the "Information") may include details of Licensee's hardware, details of the operating system(s) in use on such hardware and the profile and extent of Licensee's use of the different elements of the Software. The Foundry may use the Information to (a) model the profiles of usage, hardware and operating systems in use collectively across its customer base in order to focus development and support, (b) to provide targeted support to individual customers, (c) to ensure that the usage of the Software by Licensee is in accordance with the Agreement and does not exceed any user number or other limits on its use, and (d) to advise Licensee about service issues such as available upgrades and maintenance expiry dates. To the extent that any Information is confidential to Licensee it shall be treated as such by The Foundry. To the extent that any Information constitutes personal data for the purposes of the Data Protection Act 1998 it shall be processed by The Foundry in accordance with that Act and with The Foundry's privacy policy (see <http://www.thefoundry.co.uk/privacy/>). Licensee undertakes to make all of users of the Software aware of the uses which The Foundry will make of the Information and of the terms of The Foundry's privacy policy.

17. U.S. GOVERNMENT LICENSE RIGHTS

All Software, including all components thereof, and Documentation qualify as "commercial items," as that term is defined at Federal Acquisition Regulation ("FAR") (48 C.F.R.) 2.101, consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in FAR 12.212. Consistent with FAR 12.212 and DoD FAR Supp. 227.7202-1 through 227.7202-4, and notwithstanding any other FAR or other contractual clause to the contrary in any agreement into which this Agreement may be incorporated, a government end user will acquire the Software and Documentation with only those rights set forth in this Agreement. Use of either the Software or Documentation or both constitutes agreement by the government that all Software and Documentation are "commercial computer software" and "commercial computer software documentation," and constitutes acceptance of the rights and restrictions herein. The Software is the subject of the following notices:

* Copyright © 2001 - 2015 The Foundry Visionmongers Limited. All Rights Reserved.

* Unpublished-rights reserved under the Copyright Laws of the United Kingdom.

18. SURVIVAL.

Clause 6 and clauses 9 to 20 inclusive shall survive any termination or expiration of the Agreement.

19. IMPORT/EXPORT CONTROLS

To the extent that any Software made available under the Agreement is subject to restrictions upon export and/or re-export from the United States, Licensee agrees to comply with, and not act or fail to act in any way that would violate, the applicable international, national, state, regional and local laws and regulations, including, without limitation, the United States Foreign Corrupt Practices Act, the Export Administration Act and the Export Administration Regulations, as amended or otherwise modified from time to time, and neither The Foundry nor Licensee shall be required under the Agreement to act or fail to act in any way which it believes in good faith will violate any such laws or regulations.

20. MISCELLANEOUS

Unless Licensee is a consumer, the Agreement is the exclusive agreement between the parties concerning its subject matter and supersedes any and all prior oral or written agreements, negotiations, or other dealings between the parties concerning such subject matter. Licensee acknowledges that Licensee has not relied upon any representation or collateral warranty not recorded in the Agreement inducing it to enter into the Agreement. The Agreement may be modified only in writing. The failure of either party to enforce any rights granted under the Agreement or to take action against the other party in the event of any such breach shall not be deemed a waiver by that party as to subsequent enforcement of rights or subsequent actions in the event of future breaches. The Agreement and any dispute or claim arising out of or in connection with it or its subject matter or formation (including, unless Licensee is a consumer, non-contractual disputes or claims) shall be governed by, and construed in accordance with English Law and the parties irrevocably submit to the non-exclusive jurisdiction of the English Courts, subject to any right that a consumer may have to bring proceedings or to have proceedings brought against them in a different jurisdiction.

If The Foundry fails to insist that Licensee performs any obligation under the Agreement, or delays in doing so, that will not mean that The Foundry has waived its rights.

Unless Licensee is a consumer, Licensee agrees that The Foundry may refer to Licensee as a client or a user of the Software, may display its logo(s) for this purpose and may publish quotations and testimonials from Licensee, its directors, partners, officers or employees. The Foundry agrees to promptly cease any such use on Licensee's written request.

The Foundry and Licensee intend that each Third Party Licensor may enforce against Licensee under the Contracts (Rights of Third Parties) Act 1999 (the "Act") any obligation owed by Licensee to The Foundry under this EULA that is capable of application to any proprietary or other right of that Third Party Licensor in or in relation to the Software. The Foundry and Licensee reserve the right under section 2(3)(a) of the Act to rescind, terminate or vary this EULA without the consent of any Third Party Licensor.

Copyright © 2015 The Foundry Visionmongers Limited. All Rights Reserved. Do not duplicate.