

User Guide

VERSION 7.5v5

FOUNDRY.

Katana™ User Guide. Copyright © 2025 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 this Katana User Guide, End User License Agreement (EULA).

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 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 Foundry and/or 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 © 2025 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: Thursday, October 2, 2025

Contents

User Guide 18

Installation and Licensing 19

Katana on Windows 21
Installing on Windows 21
Licensing Katana on Windows 23
Uninstalling on Windows 26

Katana on Linux 27
Installing on Linux 28
Licensing on Linux 30
Uninstalling on Linux 33

Renderers 34

Connecting to a Renderer 35

Network Configuration 36

Python Search Path 36

Setting the Temporary File Directory 37

Managing Katana projects in Multi-Platform Environments 37

Launching Katana 40

Launching on Windows 41

Launching on Linux 42

Command-line Interface 43
Katana License Requirements for Launch Modes 44
Interactive Mode 45
Script Mode 46
Shell Mode 47
Batch Mode 47

Katana Resources 55

Environment Variables 59

What is Katana? 62

Key Concepts 64

Glossary of Katana Terms 67

User Interface 78

The Default Workspace 78

The Default Tabs 79

Menu Bar Components 80

Customizing Your Workspace 84

Adjusting Layouts 85

Saving, Loading, and Deleting Layouts 87

Managing Keyboard Shortcuts 88

Getting Help 89

Creating a Project 92

Creating, Saving, and Loading a New Project 92

Importing and Exporting a Project 94

Changing a Project's Settings 95

Assets and Asset Managers 99

Using the File Browser 99

Autosaves 102

Editing the Node Graph 107

Navigating Inside the Node Graph 107

Adding Nodes 108

Node Basics 109

Selecting Nodes 113

Connecting Nodes 114

Merging Nodes 115

Removing, Replacing, and Deleting Nodes 116

Copying, Pasting, and Cloning Nodes 117

Grouping Nodes 118
Backdrop Nodes 121
Dot Nodes 124
Advanced Display Options 124
Editing a Node's Parameters 127
Node Parameter Basics 127
Common Parameter Widgets 131
Parameter State Badges 146
Adding User Parameters 147
Widget Types 148
Widget Options 152
Conditional Behavior 153
Creating Help Text for User Parameters 157
Animation 159
Setting Keys 161
Curve Editor Overview 164
Dope Sheet Overview 186
Using the Timeline 190
Using the Scene Explorer and Scene Graph 192
Using the Scene Explorer 192
Payloads in the Scene Explorer 200
Activating and Deactivating Prims 207
Collections in the Scene Explorer 209
Using the Scene Graph 213
The Process of Generating Scene Graph Data 216
Manipulating the Scene Graph 217
Structured Scene Graph Data 221
Bounding Boxes and Good Data for Renderers 221
Proxies and Good Data for Users 222
Level-of-Detail Groups 224
Alembic and Other Input Data Formats 225
Working Sets 225
Changing What is Shown in the Viewer 231
Bookmarking a Scene Graph State and Working Sets 232
Controlling Live Rendering in the Scene Graph 234
Making Use of Different Location Types and Proxies 235
Using Assemblies and Components 236
Resolvers 237
Examples of Resolvers 239
Implicit Resolvers 239
Creating Your Own Resolvers 241

Building Your Scene 243

Adding 3D Assets 244
Adopting Alembic 245

Collections and CEL 250
CEL in the User Interface 251
Guidelines for using CEL 252
CEL in Parameters 254

Working with Attributes 254
AttributeSet Nodes 255
OpScript Nodes 262
Adding an OpScript 262
OpScript Tutorials 264

USD in Katana 274

What is USD? 276

USD Workflows - Choose your route 278

Using Native USD Workflows 280
Building USD Scenes 285
Composition Arcs - Understanding LIVRPS 287
Importing USD Data 289
Native USD Prims 291
Native USD Lights 296
Creating Native USD Cameras 299
UsdMaterialAssign Workflows 302
Using the Scene Explorer 317
Payloads in the Scene Explorer 325
Activating and Deactivating Prims 332
Collections in the Scene Explorer 334
Inspecting USD Properties Using the Attributes Tab 338
Manipulating USD Data Natively 341
Native USD Transformations 342
Bounding Boxes in USD Scenes 346
Pattern-Based Collections with USD 348
Native USD Layer Export 353
USD Debugging 361
USD Text View 362
USD Error Handling 363

Bridging between Katana Geolib and USD 364
Conversion Nodes - KatanaToUsd/ UsdToKatana 365

Pre-existing UsdIn Workflows 368
Loading in USD Assets with UsdIn 369

Setting up USD Lights 370
Setting up UsdPreviewSurface Materials 374
Loading USD Plug-ins into Katana 380

Viewing Your Scene 384

Changing the Layout 385

Selecting Within the Viewer 388

Using Flush Caches 389

Using the Viewer 389
Changing the View Position 389
Pan and Zoom 391
Selecting Faces 393
Using Manipulators in the Hydra Viewer 398
Geometry Display Options 403
The Monitor Layer in the Hydra Viewer 405
Image-Based Selection in the Monitor Layer 408
Snapping 416
Using Stereo Cameras in the Hydra Viewer 424
Subdivision and Anti-Aliasing in the Hydra Viewer 425
Live Rendering with the Hydra Viewer 427
Render Delegates in the Hydra Viewer 428
Proxies and Bounding Boxes 433
Displaying Textures in the Hydra Viewer 434
Loading USD Plug-ins into Katana 435
Setting up UsdPreviewSurface Materials 438
Setting up USD Lights 444
Changing Display Properties for Some Locations 448
Customizing the Viewport 450

Lighting Your Scene 452

Creating a Light 453

Positioning Lights 456

Light Linking 456

Getting to Grips with the GafferThree Node 457
Gaffer Object Table Overview 458
Creating a Light Using the GafferThree Node 459
Making Use of Rigs 460
Defining a Template Light Material 462
Creating a Light Filter Using the GafferThree Node 463
Linking Shadows to Specific Objects 466
Adopting Items from an Incoming Scene 466

Soloing and Muting Lights, Light Filters, and Rigs 467
Locking a Light or Rig's Transform 468
Duplicating an Item Within the Gaffer Object Table 469
Syncing the GafferThree Selection with the Scene Graph 469
Using and Overriding Look Files with GafferThree Lights 470

Lighting Tools 474
Creating Lights using Lighting Tools 475
Editing Lights Using the Lighting Tools Parameter Widget 489
Cloning Lights and Using Template Materials With Lighting Tools 500

Look Development 509

Look Development with Look Files 509
Using Look Files to Create a Material Palette 509
Using Look Files in an Asset's Look Development 511
Creating a Look File Using LookFileBake 511
Assigning a Look File to an Asset 517
Resolving Look Files 518
Overriding Look File Material Attributes 519
Activating Look File Lights and Constraints 520
Using Look Files as Default Settings 520
Bringing a Look File into the Scene Graph 522
Assigning and Unassigning a Global Look File 523
Removing a Look File from the Look Files List 524
Managing Passes in the LookFileManager 524
Overriding Look Files 525

Adding and Assigning Materials 526
Material Basics 527
Material Pipelines 532
Adding Multiple Materials 533
NMX Utility Nodes 535
Building Materials Using NetworkMaterialCreate 542
Creating Shading Networks 544
Multiple NetworkMaterials with NetworkMaterialCreate 561
Organizing Shading Networks with ShadingGroup Nodes 571
Node Parameters and Interface Controls 579
Editing Materials With The NetworkMaterialEdit Node 591
Easily Preview Sections of Your LookDev Using Material Solo 598
Network Materials 602
Creating a Network Material 603
Using a Network Shading Node 605
Creating a Network Material's Public Interface 611
Changing a Network Material's Connections 614
Editing a Network Material 615
Handling Textures 616
Texture Handling Options 616
Using Pipeline Data to Set Textures 620

Checking UVs 621
Bringing up the UV Viewer Tab 621
Navigating in the UV Viewer Tab 621
Selecting Faces 622
Adding Textures to the UV Viewer 624
Using Multi-Tile Textures 625
Changing the UV Viewer Display 626

Look Files 627
Handing off Looks from Look Development to Lighting 628
Look File Baking 628
Other Uses of Look Files 629
How Look Files Work 630
Setting Material Overrides using Look Files 631
Collections using Look Files 631
Look Files for Palettes of Materials 632
Look File Globals 632
Lights and Constraints in Look Files 632
The Look File Manager 633

Rendering Your Scene 634

Render Types 636
Render Type Availability 639

Performing a Render 642
Starting Multiple Renders 648
Multiple Live Renders with Foresight+ 654
Katana Queue 658

Configuring a Render 665
Render Dependencies 665
Rendering only Selected Locations 666
Setting up Interactive Render Filters 666
Managing Color 668

Viewing Your Renders 669
Using the Monitor Tab 669
Changing the Image Size and Position 672
Overlay Masking 673
Changing How to Trigger a Render 675
Rendering a Region of Interest (ROI) 676
Changing the Displayed Channels 680
Changing How the Alpha Channel is Displayed 680
Selecting An Output Pass 681
Viewing the Pixel Values 682
Comparing Front and Back Images 685
Toggling 2D Manipulator Display 689
Underlaying and Overlaying an Image 689

Using the Catalog Tab 690
Using the Histogram 696
Custom Render Resolutions 697
Influencing a Render 697
Controlling Live Rendering 698
Global Options 699
Setting up a Render Pass 702
Defining and Overriding a Color Output 703
Defining Outputs Other than Color 705
Defining an AOV Output 706
Previewing Interactive Renders for Outputs Other than Primary 710
Instancing 711
Rendering Instances 711
Writing EXR Metadata from Katana 716
Setting up Render Dependencies 718
Batch Mode 719
Advanced Workflow & Extensions 728
See a Nuke Comp of Your Project in Katana Using the Nuke Bridge 729
Asset Management 756
Asset Plug-ins 756
Asset Management System Plug-in API 757
Configuring the Asset Browser 771
Implementing A Custom Asset Control Widget 773
Asset Render Widget 774
Additional Asset Widget Delegate Methods 775
addAssetFromWidgetMenuItems() 776
shouldAddStandardMenuItem() 777
shouldAddFileTabToAssetBrowser() 778

getQuickLinkPathsForContext() 779
The Asset Publishing Process 780
Choosing an Asset Plug-in 780
Example Asset Plug-in 781
Retrieve and Publish 782
LiveGroups and LiveShadingGroups 786
Creating a LiveGroup 787
Editing LiveGroup Parameters 788
Loading and Reloading a LiveGroup 789
Editing the Contents of a LiveGroup 790
Making a LiveGroup Node Editable 790
Modified State of Editable LiveGroup Nodes 791
Publishing a LiveGroup 792
LiveGroup Conversion 793
Graph State Variables 794
Setting Graph State Variables 795
Inspecting Graph State Variables 796
Reading Graph State Variables 797
How Do Graph State Variables Work? 799
Scripting and Programming in Katana 799
Scripting with Python 801
Shelf Item Scripts 802
Using the Python Tab 807
Automating Procedures 810
Message Logging 811
The Op API 816
Groups, Macros, and SuperTools 834
Macros 838
SuperTools 839
Writing a SuperTool 840
Customizing GafferThree 841
Optimizing Performance 843
Geolib3-MT Configuration 844
Geolib3-MT Profiling 850
Op Cook Profiling 856
Starting and Ending a Profiling Session 857
Profiling Renders 858
Profiling Reports 858
Profiling and Optimization Guide 860

Optimize Projects Using the Performance Tab 861
Using the Python Profiler 870
Improving Your Node Graph 873
Improving Your Ops 877
Composing Concurrency-Friendly Scenes 881
Improving OpScript Performance 885

Preferences 891

Keyboard Shortcuts 907

Reference Guide 922

2D Nodes 923

Color Nodes 923
ImageBackgroundColor 923
ImageBrightness 924
ImageChannels 924
ImageClamp 927
ImageContrast 928
ImageExposure 930
ImageFade 931
ImageGain 932
ImageGamma 933
ImageInvert 934
ImageLevels 935
ImageSaturation 937
ImageThreshold 938
OCIOCDLTransform 939
OCIOColorSpace 940
OCIODisplay 941
OCIOFileTransform 942
OCIOLogConvert 944
OCIOLookTransform 944

Composite Nodes 945
ImageIn 946
ImageMerge 949
ImageOut 952
ImagePremultiply 955
ImageUnpremultiply 956
ImageZMerge 957

Filter Nodes 957
ImageBlur 957

I/O Nodes 959
ImageRead 960
ImageWrite 964

Source Nodes 973
ImageCheckerboard 974
ImageColor 975
ImageRamp 976
ImageText 980

Transform Nodes 984
ImageCrop 984
ImageOrient 985
ImagePosition 986
ImageReformat 987
ImageTransform2D 990

3D Nodes 995

Constraint Nodes 995
AimConstraint 995
BillboardConstraint 997
CameraScreenWindowConstraint 999
ClippingConstraint 1000
DollyConstraint 1002
FOVConstraint 1003
OrientConstraint 1005
ParentChildConstraint 1007
PointConstraint 1008
ReflectionConstraint 1010
ScaleConstraint 1011
ScreenCoordinateConstraint 1012

Input Nodes 1013
AttributeFile_In 1014

Lookfile Nodes 1016
LookFileBake 1016
LookFileLightAndConstraintActivator 1018
LookFileManager 1019
LookFileMaterialsIn 1020
LookFileMaterialsOut 1021
LookFileMultiBake 1022
LookFileOverrideEnable 1025
LookFileResolve 1026
UsdMaterialBake 1027

Output Nodes 1030
Render 1030

Procedural Nodes 1033
Alembic_In 1033
RendererProceduralArgs 1036

Resolve Nodes 1045
ConstraintResolve 1046
MaterialResolve 1046

Source Nodes 1046
AttributeFile_In 1046
CameraCreate 1049
CameraImagePlaneCreate 1051
CollectionCreate 1054
CoordinateSystemDefine 1055
InfoCreate 1057
LightCreate 1058
LocationCreate 1060
Material 1061
PrimitiveCreate 1064
TeapotCreate 1066

SuperTool Nodes 1067
Importomatic 1067
LookFileLightAndConstraintActivator 1068
LookFileManager 1069
LookFileMultiBake 1069

Other 3D Nodes 1072
ArnoldObjectSettings 1072
ArnoldGlobalSettings 1073
ArnoldLiveRenderSettings 1073
ArnoldOutputChannelDefine 1074
ArnoldShadingNode 1075
AttributeCopy 1076
AttributeEditor 1078
AttributeSet 1079
BoundsAdjust 1082
CameraClippingPlaneEdit 1084
ConstraintCache 1085
ConstraintListEdit 1086
FaceSetCreate 1086
GenericOp 1087
GroupMerge 1093
HierarchyCopy 1094
Isolate 1096
LightLink 1097
LightLinkEdit 1100
LightLinkResolve 1102
LightLinkSetup 1103

LightListEdit 1106
LocationGenerate 1107
LodGroupCreate 1108
LodSelect 1109
LodValuesAssign 1111
MaterialStack 1112
Merge 1112
NetworkMaterial 1115
NetworkMaterialCreate 1116
NetworkMaterialEdit 1122
NetworkMaterialInterfaceControls 1127
NetworkMaterialMultiSplice 1131
NetworkMaterialParameterEdit 1132
NetworkMaterialSplice 1133
OpResolve 1136
OpScript 1137
PrmanGlobalSettings 1142
PrmanObjectSettings 1142
PrmanOutputChannelDefine 1143
PrmanShadingNode 1144
Prune 1145
Rename 1146
RenderOutputDefine 1147
ReverseNormals 1152
ShadingGroup 1153
ShadingNodeArrayConnector 1154
ShadingNodeSubnet 1154
Transform3D 1155
TransformEdit 1156
VelocityApply 1157
ZoomToRect 1158

USD Nodes 1159

KatanaToUsd 1159

UsdActiveSet 1162

UsdAttributeSet 1163

UsdCamera 1165

UsdCollection 1168

UsdInheritSet 1171

UsdKindSet 1174

UsdLayerDefine 1175

UsdLayerExport 1176
UsdLayerExportGroup 1178
UsdLayerWrite 1180
UsdLight 1181
UsdMaterialAssign 1186
UsdMetadataSet 1187
UsdPayloadSet 1189
UsdPrimCreate 1191
UsdPrimvarSet 1194
UsdPythonWrite 1196
UsdReferenceSet 1197
UsdRelationshipSet 1199
UsdSchemaSet 1201
UsdScope 1205
UsdSpecializeSet 1206
UsdStageDefine 1209
UsdSubLayerAdd 1210
UsdToKatana 1212
UsdTransformEdit 1214
UsdTransformSet 1215
UsdVariantSet 1217
UsdXform 1219

Miscellaneous Nodes 1221

SuperTool Nodes 1221
GafferThree 1221
ImageCoordinate 1242
PonyStack 1242

Other Nodes (Misc) 1243
Backdrop 1243

DependencyMerge 1243
Dot 1244
Group 1245
GroupStack 1245
InteractiveRenderFilters 1247
LiveGroup 1247
LookFileAssign 1249
LookFileGlobalsAssign 1249
MaterialAssign 1250
NonpersistentSwitch 1251
RenderScript 1252
RendererProceduralAssign 1265
ScenegraphObjectSettings 1266
Switch 1267
Teleport 1268
TimeOffset 1268
UsdIn 1269
UsdInActivationSet 1272
VariableDelete 1273
VariableEnabledGroup 1273
VariableSet 1274
VariableSwitch 1274
ViewerObjectSettings 1275
VisibilityAssign 1279

End User License Agreement (EULA) 1281

User Guide

This manual walks you through installing, licensing, and using Katana. For information on each individual node, see the Reference Guide.

Installation and Licensing

This section guides you to the point where you have a default Katana workspace 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 7.5v5 is tested and qualified on Linux 64-bit operating system (Rocky Linux 9) and Windows 10, 11 (64-bit).
- The configurations listed below are those that Foundry have tested with Katana. Due to the constantly changing nature and wide variety of computer hardware available in the market, Foundry is unable to officially certify hardware. The list below can be used as a recommendation and does not guarantee that it meets your particular needs.
    - NVIDIA Quadro RTX 4000
    - NVIDIA Quadro RTX A4000
    - NVIDIA Quadro RTX A3000
    - NVIDIA Quadro P4000

Please download and install the latest graphics driver from the NVIDIA website.
If you encounter any issues, please contact Customer Support directly through the Support Portal at the following address: https://support.foundry.com.

- A supported renderer (see Renderers).



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 7.5v5 has dependencies on the following third-party libraries:

As part of this release this line will follow the same VFX Reference Platform as Katana 7.0; **VFX Reference Platform 2023**. Some highlights of the specific versions are included below but for a complete list, please refer to the Developer Guide.

	VFX Reference Platform 2023	**Katana 7.0**
GCC	11.2.1	11.2.1
glibc	2.28	2.28
Windows Toolset	Visual Studio 2022	Visual Studio 2022
Windows SDK	10.0.19041	10.0.22621
Python	3.10.x	3.10.10
Qt	5.15.x	5.15.2 (modified)
PyQt	5.15	5.15.9
OpenEXR	3.1.x	3.1.4
OpenSubdiv	3.5.x	3.5.0
Alembic	1.8.x	1.8.6
OpenColorIO	2.2.x	2.2.1
Boost	1.80	1.80.0
Intel TBB	2020 Update 3	2020 Update 3
OpenVDB	10.x	10.0.1
NumPy	1.23.x	1.23.5

Outside of the VFX Reference Platform, we have also updated to **USD 24.05**.

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

Katana on Windows

After installation, all Katana applications are run from either desktop icons, the **Start** menu, or from the command-line using launch modes and command-line arguments.

Katana System Requirements

Supported Operating Systems

• Windows 10,11(64-bit)



Note: Other operating systems may work with Katana, but have not been fully tested. If you have any problems with a particular operating system, please contact our support team.

Installing on Windows

To install Katana on Windows , see either:

• Installing Katana with the User Interface (UI)

• Installing Katana from the Command-Line

Installing Katana with the User Interface (UI)

To install Katana on Windows using the standard, user interface method, follow the instructions below:

1. Download the correct .zip installation file from our website at https://www.foundry.com/products/katana.
2. Extract the **.exe** installation file from the .zip file
3. Double-click on the .exe installation file to install Katana.
4. Follow the on-screen instructions. By default, Katana is installed to drive letter**:\Program Files\Katana7.5v5**.
5. Proceed to Licensing Katana on Windows .

Installing Katana from the Command-Line

To install Katana on Windows from the command-line, follow the instructions below:

1. Download the correct **.exe** installation file from our website at https://www.foundry.com/products/katana.
2. Extract the **.exe** installation file from the **.zip** file.
3. To open a command prompt window, select **Start** > **All Programs** > **Accessories** > **Command Prompt**.
4. Use the **cd** (change directory) command to move to the directory where you saved the **.exe** installation file. For example, if you saved the **.exe** installation file in **C:\Temp**, use the following command and press **Return**:
   ```
   cd \Temp
   ```
5. To install Katana, do one of the following:
   - To install Katana to the current directory and display the installation dialog, type the name of the install file without the file extension and press **Return**:
     ```
     Katana7.5v5-win-x86-release-64
     ```
   - To install Katana to a specified directory and display the installation dialog, use the **/dir** install option:
     ```
     Katana7.5v5-win-x86-release-64 /dir="C:\Katana"
     ```
   - To install Katana silently so that the installer does not prompt you for anything but displays a progress bar, enter **/silent** after the installation command:
     ```
     Katana7.5v5-win-x86-release-64 /silent
     ```
   - To install Katana silently so that nothing is displayed, enter **/verysilent** after the installation command:
     ```
     Katana7.5v5-win-x86-release-64 /verysilent
     ```
   - To install Katana, but not 3Delight, use the **/components** option:
     ```
     Katana7.5v5-win-x86-release-64 /components="!delight"
     ```



Note: This option does not stop Katana loading the 3Delight renderer at start up if it already exists on the local machine.

- You can also use a combination of install options:

```
Katana7.5v5-win-x86-release-64 /silent /dir="C:\Katana"
/components="!delight"
```



Note: By using the **/silent** or **/verysilent** install options, you agree to the terms of the Katana End User Licensing Agreement. To see this agreement, please refer to End User License Agreement (EULA).

Licensing Katana on Windows

The following licensing methods are available:

- **Activation Keys** - activation keys allow you to activate and generate your actual product license key, at a later point after purchase, on the machine for which you require the license. They are provided for both node locked and floating license, and generate the appropriate license type once installed using the product's **Licensing** dialog or online using the Activate a Product page:

  https://www.foundry.com/user/login?destination=/licensing/activate-product

- **Node Locked Licenses** - these can be used to license an application on a single machine. They do not work on different machines and if you need them to, you'll have to transfer your license.

  Node locked licenses, sometimes called uncounted licenses, do not require additional licensing software to be installed.

- **Floating Licenses** - also known as counted licenses, enable application to work on any networked client machine. The floating license is put on the server and is locked to a unique number on that server.

  Floating licenses on a server requires additional software to be installed on the server to manage the licenses and give them out to the client stations that want them. This software is called the Foundry Licensing Tool (FLT) and can be downloaded at no extra cost from our website.

The following instructions run through the basic options for the first two licensing methods, but can find a more detailed description in the *Foundry Licensing Tools (FLT) User Guide* available on our website:

https://www.foundry.com/licensing

To use Katana, you need either a node locked license or a floating license and a server running the Foundry Licensing Tool (FLT). Katana uses RLM licensing, and the default local RLM location is:
`C:\ProgramData\The Foundry\RLM`

Obtaining Licenses

To obtain a license, you'll need your machine's System ID (sometimes called Host ID or rlmhostid). Just so you know what a System ID number looks like, here's an example: 000ea641d7a1.



Note: Bear in mind that, for floating licenses, you'll need the System ID of the license server, not the machines on which you intend to run the application.

There are a number of ways you can find out your machine's System ID:

- Launch the application without a license, click **Status** in the bottom-left of the dialog, and then scroll down the error report until you see your System ID.
- Download the Foundry License Utility (FLU) from https://www.foundry.com/licensing and run it. Your System ID is displayed.
- Download the Foundry Licensing Tools (FLT) free of charge from https://www.foundry.com/licensing and then run **C:\Program Files\TheFoundry\LicensingTools7.0\ Foundry License Utility.exe**

When you know your System ID, you can request a license for Foundry products:

- from the Foundry's Sales Department at sales@foundry.com.
- by registering your interest in the product by filling out the form on https://www.foundry.com/user/register?request_uri=/products/4/trial/interactive.

Installing Licenses

When you start the application before installing a license, a Licensing dialog displays an error informing you that no license was available. The installation process is dependent on what type of license you requested:

- **License file** - if you requested a license file, typically **foundry.lic**, this option allows you to browse to the file location and install it automatically. See To Install a License from Disk for more information.
- **Activation Key or license text** - if you requested an Activation Key or license by email, this option allows you to paste the key or license text into the Licensing dialog, which then installs the license in the correct directory. See To Install an Activation Key or License Text for more information.
- **A floating license** - if you requested a floating license to supply licenses to multiple client machines, this option allows you to enter the server address that supplies the client licenses. See To Install a Floating License for more information.



Note: You must install a floating license and additional software on the license server to use this option.

To Install a License from Disk

1. Save the license file to a known location on disk.
2. Launch the application.

   The **Licensing** dialog displays.
3. Click **Install License** to display the available license installation options.
4. Click **Install from Disk**.
5. Browse to the location of the license file.
6. Click **Open** to install the license automatically in the correct directory.

   Once you click **Open**, you should see the **Licenses Updated** message in the **Licensing** dialog and you are presented with a **Launch** option.
7. Click **Launch** to open Katana.

To Install an Activation Key or License Text

1. Launch the application.

   The **Licensing** dialog displays.
2. Click **Install License** to display the available license installation options.
3. Click **Activation Key / License Text** and then either:
   - Enter the Activation Key string in place of Insert Activation Key Here. A license key typically looks something like this:

     `katana-0101-77d3-99bd-a977-93e9-8035`

     OR
   - Copy the license text and paste it over the *Copy/Paste license text here* text. License text typically looks something like this:

     ```
     LICENSE foundry katana_i 2013.0929 29-sep-2013 uncounted
     hostid=000a957bfde5 share=h min_timeout=30 start=29-sep-2012
     issued=29-sep-2012 disable=VM _ck=da32d7372f sig="60P0450MJRP97E3DP
     B42C99Y5UAPRMEMGNQ39PG22H4WGH3WFK2KPTXFWJTYR0GYASJBXC0PU8"
     ```
4. Click **Install**.

   The license is automatically installed on your machine in the correct directory. Once you click **Install**, you should see the **Licenses Updated** message in the **Licensing** dialog and you are presented with a **Launch** option.

5. Click **Launch** to open Katana.



Note: Activation Keys require an internet connection. If you access the internet through a proxy server and cannot connect to the activation server, you may get an error dialog prompting you to either:

- Click **Use Proxy** to enter the proxy server name, port number, username, and password. This enables the application to connect to the activation server and obtain a license. The license is then installed automatically, or
- Click on the web link in the dialog and use the System ID (also known as hostid) provided to manually activate and install a license.

To Install a Floating License

If you requested a floating license from Foundry, you will receive your license key (**foundry.lic**) in an email or internet download. You should also receive the Foundry License Tools (FLT) application. Once you've installed the FLT, you can access the Foundry Licensing Utility (FLU) to help you install the license key on the license server machine. The server manages licenses for the client machines on your network.

1. From the **Start** menu, navigate to **All Programs** > **The Foundry** > **FLT 7.1v1**, then right-click on the "Foundry License Utility" and select "Run as Administrator".
2. Run the FLU on the client machine and paste the following line into the License Install tab:

   HOST <server name> any 4101

   replacing <server name> with the hostname of your server, for example:

   HOST red any 4101

Further Reading

There is a lot to learn about licenses, much of which is beyond the scope of this manual. For more information on licensing, displaying the System ID number, setting up a floating license server, adding new license keys and managing license usage across a network, you should read the *Foundry Licensing Tools (FLT) User Guide*, which can be downloaded from our website, https://www.foundry.com/licensing.

Uninstalling on Windows

To uninstall Katana on Windows , there are a few things you need to do:

1. Navigate to **Start** > **All Programs** > **The Foundry** > **Katana7.5v5** and select **Uninstall**.

   The **Katana Uninstall** dialog displays.
2. Click **Yes** to uninstall Katana.
3. Delete, rename, or move your **.katana** folder, if it exists.

   The **.katana** folder is usually found under the directory pointed to by the **HOME** environment variable. If this variable is not set, which is common, the **.katana** directory is under the folder specified by the **USERPROFILE** environment variable, which is generally one of the following:

   ```
   drive letter:\Documents and Settings\login name\
   drive letter:\Users\login name\
   ```

   To find out if the **HOME** and **USERPROFILE** environment variables are set, and where they are pointing at, enter **%HOME%** or **%USERPROFILE%** into the address bar in Windows Explorer. If the environment variable is set, the folder it's pointing at is opened. If it's not set, you get an error.
4. Delete, rename, or move your cached files, which reside in the following directory by default:

   `~\AppData\Local\Temp\katana*`

   Where **~** is equal to **%HOME%** or **%USERPROFILE%** as detailed above.



Note: If you specified an alternate directory using the **KATANA_TMPDIR** environment variable, purge those files as well as the default location. See **Help** > **Developer Guide** for more information.

Katana on Linux

After installation, all Katana application are run from either desktop icons, the browser, or from the terminal using launch modes or arguments.

Katana System Requirements

Qualified Operating Systems

• Linux 64-bit operating system (Rocky Linux 9)



Note: Other operating systems may work with Katana, but have not been fully tested. If you have any problems with a particular operating system, please contact our support team.

Installing on Linux

To install Katana on Linux, see either:

- Installing Katana from the Terminal, or
- Installing Katana Remotely from the Terminal.

Installing Katana from the Terminal

To install Katana on Linux from the terminal, follow the instructions below:

1. Download the correct **.tgz** installation file from our website at https://www.foundry.com/products/katana.
2. Move the **.tgz** file into a temporary folder.
3. Extract and decompress the **.tgz** file inside the temporary folder.
   ```
   tar xvf Katana<version>-linux-x86-release-64.tgz
   ```
   This gives you an installer file.
4. Start the install script:
   ```
   ./install.sh
   ```
   The install script supports several command-line options:
   - **-h** or **--help** - displays the available options.
   - **--accept-eula** - automatically accepts the EULA without displaying it.
   - **--path** or **--katana-path** - specifies where Katana is installed and accepts the EULA without displaying it. For example, 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
     ```



Note: By installing Katana with the **--accept-eula** or **--path** options, you agree to the terms of the End User Licensing Agreement. To see this agreement, please refer to https://www.foundry.com/eula.

   - **--3delight-path** - specifies where 3Delight is installed.
   - **--no-3delight** - disables the automatic installation of 3Delight.
5. Read and acknowledge the End User License Agreement (EULA) by pressing **Y** at the end of the text.

Tip: If you've already read and agreed to the terms of the EULA, you can skip to the end of the text by pressing **Q**.

6. Enter the installation directory for Katana or press **Enter** to accept the default install directory.
7. If you didn't add a license key during the installation, do that now using the instructions in Licensing on Linux. Otherwise, proceed to Launching on Linux.

Additionally, you can install Katana silently by simply unzipping the installer file. This creates the properly formed Katana directory tree in the current directory.

Installing Katana Remotely from the Terminal

To install Katana on Linux remotely from the terminal, follow the instructions below:

1. Download the correct **.tgz** installation file from our website at https://www.foundry.com/products/katana.
2. Extract the installer from the **.tgz** archive with the following terminal command:
   ```
   tar xvzf Katana7.5v5-linux-x86-release-64.tgz
   ```
   This gives you an installer file.
3. Use the following terminal command to log in to your render machine as root:
   ```
   ssh root@render_machine
   ```
   Replace *render_machine* with the name of your render node.
4. Make a directory to install Katana to:
   ```
   mkdir /usr/local/Katana7.5v5
   ```
5. Copy the installer file from the machine on which you downloaded it to your render machine with a command like:
   ```
   scp root@download_machine: /tmp/Katana7.5v5-linux-x86-release-64-installer
   root@render_machine: /usr/local/Katana7.5v5/
   ```
   Replace *download_machine* with the name of the machine to which you downloaded the installer file, and *render_machine* with the name of your render node.
6. Unzip the installer file to unpack its contents into your Katana directory:
   ```
   cd /usr/local/Katana7.5v5
   unzip Katana7.5v5-linux-x86-release-64-installer
   ```
7. Repeat steps 3-6 for each render machine.

Licensing on Linux

The following licensing methods are available:

- **Activation Keys** - activation keys allow you to activate and generate your actual product license key, at a later point after purchase, on the machine for which you require the license. They are provided for both node locked and floating license, and generate the appropriate license type once installed using the product's **Licensing** dialog or online using the Activate a Product page:

  https://www.foundry.com/user/login?destination=/licensing/activate-product

- **Node Locked Licenses** - these can be used to license an application on a single machine. They do not work on different machines and if you need them to, you'll have to transfer your license.

  Node locked licenses, sometimes called uncounted licenses, do not require additional licensing software to be installed.

- **Floating Licenses** - also known as counted licenses, enable application to work on any networked client machine. The floating license is put on the server and is locked to a unique number on that server.

  Floating licenses on a server requires additional software to be installed on the server to manage the licenses and give them out to the client stations that want them. This software is called the Foundry Licensing Tool (FLT) and can be downloaded at no extra cost from our website.

The following instructions run through the basic options for the first two licensing methods, but can find a more detailed description in the *Foundry Licensing Tools (FLT) User Guide* available on our website:

https://www.foundry.com/licensing

To use Katana, you need either a node locked license or a floating license and a server running the Foundry Licensing Tool (FLT). Katana uses RLM licensing, and the default local RLM location is:
`/usr/local/foundry/RLM`

Obtaining Licenses

To obtain a license, you'll need your machine's System ID (sometimes called Host ID or rlmhostid). Just so you know what a System ID number looks like, here's an example:

000ea641d7a1.



Note: Bear in mind that, for floating licenses, you'll need the System ID of the license server, not the machines on which you intend to run the application.

There are a number of ways you can find out your machine's System ID:

- Download the Foundry Licensing Tools (FLT) free of charge from https://www.foundry.com/licensing and then run the following command in a terminal shell:

```
/usr/local/foundry/LicensingTools7.0/bin/systemid
```

- Download the Foundry Licensing Utility (FLU) free of charge from https://www.foundry.com/licensing and then run the following command in a terminal shell:

```
./FoundryLicenseUtility -i
```

When you know your System ID, you can request a license for Foundry products:

- from the Foundry's Sales Department at sales@foundry.com
- by registering your interest in the product by filling out the form on https://www.foundry.com/user/register?request_uri=/products/4/trial/interactive.

Installing Licenses

When you start the application before installing a license, a Licensing dialog displays an error informing you that no license was available. The installation process is dependent on what type of license you requested:

- **License file** - if you requested a license file, typically **foundry.lic**, this option allows you to browse to the file location and install it automatically. See To Install a License from Disk for more information.
- **Activation Key or license text** - if you requested an activation key or license by e-mail, this option allows you to paste the key or license text into the Licensing dialog, which then installs the license in the correct directory. See To Install an Activation Key or License Text for more information.
- **A floating license** - if you requested a floating license to supply a license to multiple client machines, this option allows you to enter the server that supplies the client licenses. See To Install a Floating License for more information.



Note: You must install a floating license and additional software on the license server to use this option.

To Install a License from Disk

1. Save the license file to a known location on disk.
2. Launch the application.

The **Licensing** dialog displays.

3. Click **Install License** to display the available license installation options.
4. Click **Install from Disk**.
5. Browse to the location of the license file.
6. Click **Open** to install the license automatically in the correct directory.

   Once you click **Open**, you should see the **Licenses Updated** message in the **Licensing** dialog and you are presented with a **Launch** option.
7. Click **Launch** to open Katana.

To Install an Activation Key or License Text

1. Launch the application.

   The **Licensing** dialog displays.
2. Click **Install License** to display the available license installation options.
3. Click **Activation Key / License Text** and then either:
   - Enter the Activation Key string in place of Insert Activation Key Here. A license key typically looks something like this:

     `katana-0101-77d3-99bd-a977-93e9-8035`

     OR
   - Copy the license text and paste it over the Copy/Paste license text here string. License text typically looks something like this:

     ```
     LICENSE foundry katana_i 2013.0929 29-sep-2013 uncounted
     hostid=000a957bfde5 share=h min_timeout=30 start=29-sep-2012
     issued=29-sep-2012 disable=VM _ck=da32d7372f sig="60P0450MJRP97E3DP
     B42C99Y5UAPRMEMGNQ39PG22H4WGH3WFK2KPTXFWJTYR0GYASJBXC0PU8"
     ```
4. Click **Install**.

   The license is automatically installed on your machine in the correct directory. Once you click **Install**, you should see the **Licenses Updated** message in the **Licensing** dialog and you are presented with a **Launch** option.
5. Click **Launch** to open Katana.

   Activation keys require an internet connection. If you access the internet through a proxy server and cannot connect to the activation server, you may get an error dialog prompting you to either:
   - Click **Use Proxy** to enter the proxy server name, port number, username, and password. This enables the application to connect to the activation server and obtain a license. The license is then installed automatically, or
   - Click on the web link in the dialog and use the System ID (also known as hostid) provided to manually activate and install a license.

To Install a Floating License

If you requested a floating license from Foundry, you will receive your license key (**foundry.lic**) in an email or internet download. You should also receive the Foundry License Tools (FLT) application. Once you've installed the FLT, you can access the Foundry Licensing Utility (FLU) to help you install the license key on the license server machine. The server manages licenses for the client machines on your network.

1. Open a terminal and browser to the directory location where the FLU is located.
2. Run the following command as a Root user:
   `./FoundryLicenseUtility -c 4101@<server_name>`
   replacing <server_name> with the hostname of your license server, for example:
   `./FoundryLicenseUtility -c 4101@red`
   The FLU creates an RLM license, and this installs a client license file on disk enabling the machine to get a license from the server.

Further Reading

There is a lot to learn about licenses, much of which is beyond the scope of this manual. For more information on licensing Katana, displaying the System ID number, setting up a floating license server, adding new license keys, and managing license usage across a network, you should read the Foundry Licensing Tools (FLT) User Guide, which can be downloaded from our website at https://www.foundry.com/licensing.

Uninstalling on Linux

To uninstall Katana on Linux, there are a few things you need to do:

1. Navigate to **/usr/local/** and delete the **Katana 7.5v5** folder.
2. Delete, rename, or move your **.katana** folder, if it exists.
   The **.katana** folder is found in your home directory, by default:
   `/home/<login name>/.katana`
3. Delete, rename, or move your cached files, which reside in the following directory by default:
   `/var/tmp/katana`



Note: If you specified an alternate directory using the **KATANA_TMPDIR** environment variable, purge those files as well as the default location. See Environment Variables for more information.

Renderers

3Delight Renderer

Katana comes bundled with 3Delight: a uni-directional path-tracer designed to withstand the high demands of production rendering.



Article: For more information about downloading and installing the 3Delight renderer and plug-in for Katana, please refer to the Knowledge Base article: 3Delight for Katana



Note: The Katana Installer provides an option for installing 3Delight with Katana.

Third-Party Renderers

Pixar's RenderMan, Solid Angle's Arnold, and Maxon's Redshift are supported by plug-ins supplied directly by those companies.

For Pixar's RenderMan, please contact Pixar to get RenderMan for Katana (also called RfK). You also need to install the relevant version of the RenderMan renderer (RenderMan Pro Server).

For Arnold, please contact Solid Angle to get Arnold for Katana (also called KtoA). This includes both the Arnold renderer as well as the Katana plug-in.

For Redshift, please contact Maxon to get Redshift for Katana.

All queries regarding third-party plug-ins should be directed to the relevant provider.

Connecting to a Renderer

Using 3Delight

The 3Delight renderer and plug-in are bundled with Katana. No additional configuration is necessary after installation.



Article: For more information about downloading and installing the 3Delight renderer and plug-in for Katana, please refer to the Knowledge Base article: 3Delight for Katana

Using Other Renderers

Katana is designed to be renderer agnostic. A number of third-party renderer plug-ins are available, supporting renderers such as RenderMan, Arnold, V-Ray and 3Delight.

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** > **Developer Guide** menu option inside Katana.

Katana uses the **KATANA_RESOURCES** environment variable to find the renderer plug-ins it needs.

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=dl

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, **dl** is used by default.



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

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 behavior 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.

Setting the Temporary File Directory

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.

Managing Katana projects in Multi-Platform Environments

When sharing Katana projects across different machines, it is sometimes necessary to adjust the format of file paths for different operating systems or to account for different folder structures.

To make a Katana project portable across multiple platforms, file paths should be set up so they are relative and not dependent on a system-specific folder structure. There are three ways to do this:

- Using parameter expressions
- Using an environment variable
- Relative file paths without using an expression

Using Parameter Expressions

Python Expressions

Relative file paths can be set up by using Python parameter expressions. For example, right-click on a filename parameter, choose 'Expression' as the Value Mode and enter an expression:

```
project.dir + '/textures/testFile.png'
```

project.dir will then be resolved to the directory of the Katana project file. This is also valid:

```
path.join(project.dir, '/textures/testFile.png')
```



Tip: See the Katana Developer Guide for more information on Python expressions.

Reference Expressions

A reference expression is a form of parameter expression that can be evaluated without the overhead of a Python interpreter.

As of Katana 3.6, parameter reference expressions support concatenation using the + operator. For example:

```
=^/user.page + '_regionExtra'
```



Tip: See the Katana Developer Guide for more information on reference expressions.

Using an Environment Variable

Alternatively, you can set an environment variable to point to the system specific root folder. To evaluate the variable in your parameter, there are two options:

- Use a parameter expression. For example:

```
getenv("OS_PATH", tmpDir) + '/example/file/path'
```

- Some nodes like Alembic_In also support the use of environment variables in a constant value for a file path parameter. For example:

```
${OS_PATH}/example/file/path
```



Note: This is not supported for every node type, in this case please use the first option of evaluating the environment variable via an expression.

Using a Relative File Path Without Using an Expression

Another option is to make use of relative file paths without using an expression. In this case you should specify your file paths relative to the project directory.

If you are launching Katana from the command line or use a bash or batch script, you can use the `cd` command to change the working directory for the environment you are launching Katana in.

For example, if your project is located here:

```
C:/Users/username/Documents/Katana/Projects
```

and you want to write relative path references to files located here:

```
C:/Users/username/Documents/Katana/Projects/textures
```

you should set your working directory to the location of your Katana project file using the `cd` command:

```
cd C:/Users/username/Documents/Katana/Projects
```

Now you can write file paths relative to your working directory:

```
/textures/testFile.png
```

If your current working directory is specified incorrectly, the texture file paths cannot be resolved.

For example, setting your working directory to:

```
cd C:/Users/username/Documents/Katana
```

resolves the texture file relative to:

```
C:/Users/username/Documents/Katana/textures/testFile.png
```

Tip: If you are launching Katana from a command line, you can type the first few characters of a directory or file name, then press **Tab** to autocomplete the file or directory path. This may help to ensure your working directory is set correctly.
You can also use the `dir` (Windows) or `ls` (Linux) commands to list all files and directories in the current or specified directory.

If you are using a Python script to launch Katana, set the root using the Python `os.chdir` command similar to the following:

```
import os
from os.path import expanduser

project_directory = 'Projects'

os.chdir(os.path.join(expanduser('~'), 'Documents', 'Katana', project_
directory))
```

The working directory is now set to:

```
C:/Users/username/Documents/Katana/Projects
```



Article: For more information on how to set up a launcher script:
Q100242: Creating a Katana launcher script for Windows
Q100272: Creating a Katana launcher script for Linux

Launching Katana

This chapter walks you through how to get Katana up and running on your platform.

Launching on Windows - Launching Katana on Windows

Launching on Linux- Launching Katana on Linux

Command-line Interface - Katana has a number of command line arguments to tailor its operation.

Katana Resources - Using the KATANA_RESOURCES environment variable.

Environment Variables - Setting and checking environment variables.

Launching on Windows

To launch the application on Windows , do one of the following:

- Double-click the Katana icon on the Desktop.
- Navigate to **Start** > **All Programs** > **The Foundry** > **Katana7.5v5** and select **Katana7.5v5**.
- Using a command prompt, navigate to the Katana application directory (by default, **\Program Files\Katana7.5v5**) and enter:
    - **bin\katanaBin.exe**

If you already have a valid license, the graphical interface appears, and a command-line window opens. If you don't have a license or haven't installed one yet, proceed to Licensing Katana on Windows .

You can also specify a Katana scene to load when Katana is launched. To do this:

1. Open the command prompt.
2. Navigate to the directory where you installed Katana.
3. Enter:
   `bin\katanaBin.exe C:\<yourDirectory>\<yourScene>.katana`
   Specifying the scene and the directory where it's located tells Katana to open this file when it launches.

There are a number of different modes for launching Katana:

- Interactive mode is the default mode. It requires no additional command-line arguments, and is 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.

In addition to the different modes, you can also set startup scripts to run on open in files named **init.py**, located in a **Startup** folder, under the path defined in the **KATANA_RESOURCES** environment variable. Alternatively, you can use a startup script in the form of an **init.py** file placed in the **.katana** folder in your **HOME** directory. These startup scripts can be run regardless of the launch mode you choose.

For information on starting Katana in the other launch modes, see Command-line Interface.

Launching on Linux

To launch the application on Linux, do one of the following:

- Double-click the Katana icon on the Desktop.
- Open the Katana application directory (by default, **/usr/local/Katana7.5v5**) and double-click the Katana icon.
- Using a terminal, navigate to the Katana application directory and enter:

```
./katana
```

If you already have a valid license, the graphical interface appears. If you don't have a license or haven't installed one yet, proceed to Licensing Katana on Linux.

You can also specify a Katana scene to load when Katana is launched. To do this:

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

There are a number of different modes for launching Katana:

- Interactive mode is the default mode. It requires no additional command-line arguments, and is 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.
- Profiling mode runs Katana with a special version of the Geolib3 Runtime that implements profiling.

In addition to the different modes, you can also set startup scripts to run on open in files named **init.py**, located in a **Startup** folder, under the path defined in the **KATANA_RESOURCES** environment variable. Alternatively, you can use a startup script in the form of an **init.py** file placed in the **.katana** folder in your **HOME** directory. These startup scripts can be run regardless of the launch mode you choose.

For information on starting Katana in the other launch modes, see Command-line Interface.

Command-line Interface

Katana's launch behavior and mode of operation can be controlled by passing command-line arguments.

Global Arguments

These arguments can be passed regardless of the selected launch mode, see Launch Modes.

Argument	Description
-h, --help	Displays a list of command-line arguments.
--asset ASSETID	Loads the Katana project with the given asset ID or path.
--ocio PATH	Uses the OpenColorIO configuration file at the given path.
--profile	Runs Katana in profiling mode. See Op Cook Profiling.
--force-profile	Runs Katana in profiling mode and start profiling immediately.
--profiling-dir DIR	Sets the directory where profiling files, if any, are written.
-V, --verbose	The level of verbosity of logging informational messages. Defaults to 1. Set to 0 to suppress most informational messages.

Launch Modes

Katana normally operates in interactive (GUI) mode. Katana can be launched in a specific launch mode by using one of the following command-line arguments.

Mode Name	Argument	Description
Interactive	No flags	Runs Katana with the standard GUI.
Batch	--batch	Runs Katana without a GUI to render the output of a specific node in the **Node Graph**. Batch mode supports further command-line

		arguments, see Batch Mode for more information.
Script	--script PATH	Runs Katana without a GUI, and executes the specified Python script at the given path.
Shell	--shell	Runs Katana without a GUI, and allows Python commands to be run interactively.

Katana License Requirements for Launch Modes

There are two types of license for Katana, interactive licenses (katana_i) and render licenses (katana_r). The type of license required depends on the mode Katana is launched in.

Katana Interactive Mode

Running Katana in Interactive (GUI) Mode requires a Katana interactive license (katana_i). This is the standard launch mode for Katana and requires no arguments when running Katana using a Command Prompt or a Terminal.

Katana interactive mode is typically used by artists or technical developers to set up or light projects, and to render using the **Preview**, **Live** or **Disk Render** commands.

For more information on Katana Interactive Mode, see Interactive Mode.



Note: Using a Katana interactive license does not allow you to access the terminal modes - batch mode, script mode or shell mode - these require a Katana render license.

Katana Terminal Modes

Katana has three terminal modes - batch mode, script mode and shell mode. All terminal modes require a Katana render license (katana_r).

- Batch Mode - used to render a Katana scene from a terminal.
- Script Mode - used to execute Python scripts in Katana's Python environment.
- Shell Mode - used to expose Katana's Python interpreter in the terminal shell.



Note: Licenses are used on a one-per-host machine basis. If you are rendering your Katana scenes on your Render Farm, using either batch mode or script mode, then each render machine will require a katana_r license.



Note: For more information on Installation and Licensing, see Installation and Licensing.

Interactive Mode

Interactive mode is the default mode, requiring no additional command-line arguments. It also loads additional modules, such as the **ScenegraphManager**. Interactive is the only mode that launches Katana with the GUI.

To start Katana in 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 on Windows or Licensing on Linux for more on this.

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 specify an asset ID using the **--asset** flag, to resolve and open a file from your asset management system. The **--asset** flag takes a single argument, which is the asset ID to resolve. For example:
`./katana --asset=mock:///show/shot/name/version`



Note: The format of the asset ID itself is dependent on your asset management system, and the file you attempt to resolve must be a valid Katana scene.



Note: The --asset flag also applies to Katana's Batch mode.

For more on Katana's Asset API see Asset Management System Plug-in API.

Script Mode

Script mode allows you to execute Python scripts in Katana's Python environment. Script mode requires the **--script** flag, followed by a single argument specifying the script you want to run. This launch mode is most useful for testing. You can import most Katana modules, and perform tasks such as loading Katana scenes, changing some parameters, and rendering.

For example, to start Katana in Script mode using a script named **yourScript.py**:

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

```
./katana --script /yourDirectory/yourScript.py
```

To open a scene and start rendering from the scene's Render node, open the following Python script in Script mode:

```
    import NodegraphAPI
    from Katana import KatanaFile
    from Katana import RenderManager

    def messageHandler( sequenceID, message ):
        print message

    yourKatanaScene = "/yourDirectory/yourFile.katana"
    KatanaFile.Load( yourKatanaScene ) # Loading scene
/yourDirectory/yourFile.katana
    RenderNode = NodegraphAPI.GetNode('Render') # Getting Render node
    RenderManager.StartRender(
        node=RenderNode, # Starting render
        hotRender=True,
        frame = 1,
```

```
        asynch = False,
        interactive = False,
        asynch_renderMessageCB = messageHandler
    )
```

Shell Mode

Shell mode exposes Katana's Python interpreter in the terminal shell. Shell mode requires the **--shell** flag, and no arguments. All of the modules available in the **Python** tab in Katana are available in Shell mode.

To start Katana in Shell mode:

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

```
./katana --shell
```

Batch Mode

Batch mode allows you to render sequences of frames from a Katana scene all at once. It is started through a command line, where you specify the file path, frame range and any other necessary options.



Note: You will only be able to access terminal modes, including Batch Mode, if you have a Katana render license (katana_r). If you're a student, you can access one for free.

Batch mode is useful if you have a large number of frames to render as it will render out each individual file in the background. You can continue working on a Katana scene file whilst it is being batch rendered as the command uses the last saved version.

Before starting a batch render, ensure the render settings and the render flag are all set up correctly in Katana. To set the render flag, select the node you wish you render from and press **V** on the keyboard. The render flag can be determined through the command line, however setting it up beforehand simplifies the string needed to run Batch mode and minimizes any room for error.



Note: When you specify the Image Filename for the output render, ensure you use one or more hashes as they will be replaced by the frame number in your rendered file name. For example: fileName_<aov>_###.<ext>

Start a Batch Render

Windows

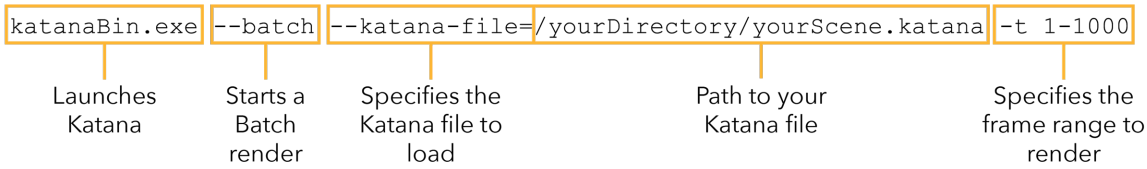
1. Open the Command Prompt.
2. Navigate to the directory where you have Katana installed using the cd command, for example:

```
cd C:\Program Files\Katana3.2v1\bin
```

3. Enter the following command to start a batch render:

```
katanaBin.exe --batch --katana-file=C:\yourDirectory\yourScene.katana
-t 1-1000
```

Where:



4. Press **Enter** to start the render.

You can add more arguments to the command. For example, use **--render-node** to specify the node you would like to render from if you haven't set your render flag in the Katana scene or if you would like to change it:

```
katanaBin.exe --batch --katana-file=C:\yourDirectory\yourScene.katana --
render-node=renderHere -t 1-1000
```

Linux

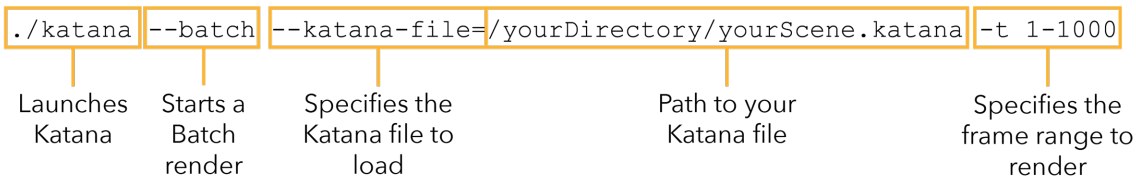
1. Open a Terminal.
2. Navigate to the directory where you have Katana installed using the cd command, for example:

```
cd /opt/foundry/katana
```

3. Enter the following command to start a batch render:

```
./katana --batch --katana-file=/yourDirectory/yourScene.katana -t 1-
1000
```

Where:



4. Press **Enter** to start the render.

You can add more arguments to the command. For example, use **--render-node** to specify the node you would like to render from if you haven't set your render flag in the Katana scene or if you would like to change it:

```
./katana --batch --katana-file=/yourDirectory/yourScene.katana --render-node=renderHere -t 1-1000
```

Here is a full list of command line options for Batch Mode:

Option	Usage

--katana-file	Specifies the Katana recipe to load.
	
	Syntax:
	`--katana-file=<filename>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty`

--asset	Specifies the asset ID to resolve.
	
	Syntax:
	`--asset=<asset ID>`
	
	Example:
	`./katana --asset=mock:///show/shot/name/version`

-t or --t	Specifies the frame range to render.
	
	Syntax:
	`-t <frame range>`
	
	OR
	`--t=<frame range>`

Option	Usage
	Where <frame range> can take the form of a range (such as 1-5) or a comma separated list (such as 1,2,3,4,5). These can be combined, for instance: 1-3,5, which would render frames 1, 2, 3, and 5. Example: `./katana --batch --katana-file=/tmp/test.katana **--t=1-5,8** --render-node=beauty`

--var	Sets the value of an existing Graph State Variable. This command-line option can be specified multiple times to override the values of multiple Graph State Variables.
	
	Syntax:
	`--var <GSV name>=<GSV value>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1 --var Shot=Sh1 --var timeOfDay=night --var variant=B --render-node=beauty`

--threads2d	Specifies the number of additional processors within the application. An additional processor is also used for Katana's main thread.
	
	This means that Katana uses 3 processors when **--threads2d=2**.
	
	Syntax:
	`--threads2d=<num threads>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000 **--threads2d=2** --render-node=beauty`

--threads3d	Specifies the number of simultaneous threads the renderer uses.
	
	Syntax:
	`--threads3d=<num threads>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000 **--threads3d=8** --render-node=beauty`

Option	Usage
--render-node	Specifies the Render node from which to render the recipe. Syntax: `--render-node=<node name>` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000` **`--render-node=beauty`**
--render-internal-dependencies	Allows any render nodes that don't produce **asset** outputs to be rendered within a single katana **--batch** process. Asset outputs are determined by asking the current asset plug-in if the output location is an assetId, using **isAssetId()**. The default **file** asset plug-in that ships with Katana classes everything as an asset. So at present it is not possible to render any dependencies within one katana **--batch** command without customizing the asset plug-in.

--crop-rect	Specifies which part of an image to crop. The same cropping area is used for all renders.
	
	Syntax:
	`--crop-rect="(<left>,<bottom>,<width>,<height>)"`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000`
	`--render-node=beauty` **`--crop-rect="(0,0,256,256)"`**

--setDisplayWindowToCropRect	Sets the display image to the same size as the crop rectangle set by **--crop-rect**.
--tile-render	Used to render one tile of an image divided horizontally and vertically into tiles. For instance, using **--tile-render=1,1,3,3** splits the image into 9 smaller images (or tiles) in a 3x3 square and then renders the middle tile as the index for tile renders starts at the bottom-left corner with 0,0. In the case of 3x3 tiles, the indices are: 0,2 1,2 2,2

Option	Usage
	0,1 1,1 2,1 0,0 1,0 2,0 The results are saved in the same location as specified by the RenderOutputDefine node but with a tile suffix. For instance: tile_1_1.beauty.001.exr Syntax: `--tile-render=<left_tile_index>, <bottom_tile_index>, <total_tiles_width>, <total_tiles_height>` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=0,0,2,2`** `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=0,1,2,2`** `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=1,0,2,2`** `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=1,1,2,2`**

--tile-stitch	Used to assemble tiles rendered with the **--tile-render** flag into a complete image.
	
	When stitching, you must still pass the **--tile-render** argument, with the number of x and y tiles, so that the stitch knows how many tiles to expect, and their configuration.
	
	Syntax:
	`--tile-render=<left_tile_index>, <bottom_tile_index>, <total_tiles_width>, <total_tiles_height> --tile-stitch`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana -`

Option	Usage
	`-t=1-1000 --render-node=beauty --tile-render=0,0,2,2` **`--tile-stitch`**
--tile-cleanup	Used to clean up transient tile images. Can be used in conjunction with **--tile-stitch** to assemble a complete image, and remove transient tiles in a single operation. When using **--tile-cleanup** you must still pass the **--tile-render** argument with the number of x and y tiles, so that cleanup knows how many tiles to remove. Syntax: `--tile-render=0,0,<total_tiles_width>,<total_tiles_height> --tile-cleanup` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty --tile-render=0,0,2,2 --tile-stitch` **`--tile-cleanup`**

--prerender-publish	In Batch mode, it executes the **Pre-Render Publish Asset** action on the outputs but doesn't render images.
	
	The value specifies the filename for dumping render pass information.
	
	Note: This can be used together with `--versionup`.
	
	Syntax:
	`--prerender-publish=<pass info>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--prerender-publish`**`=/tmp/pass_info.xml`

--make-lookfilebake-scripts	Used to write out a number of Python files that can be executed in Batch mode to write look files. Syntax:

Option	Usage

	`--make-lookfilebake-scripts=<script directory>`
	
	Example:
	`./katana --batch --katana-file=/tmp/bake.katana --t=1`
	`--make-lookfilebake-scripts=/tmp/bake_scripts`
	`./katana --script /tmp/bake_scripts/preprocess.py`
	`./katana --script /tmp/bake_scripts/lf_bake_default.py`
	`./katana --script /tmp/bake_scripts/postprocess.py`

--postrender-publish	In Batch mode, it executes the **Post-Render Publish Asset** action on the outputs but doesn't render images.
	
	The value specifies the filename for dumping render pass information.
	
	  **Note:** This can be used together with `--versionup`.
	
	Syntax:
	`--postrender-publish=<pass info>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--postrender-publish`**`=/tmp/pass_info.xml`

--versionup	Used to specify that you want to version up assets when publishing to the asset management system. Syntax: `--versionup` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--versionup`**
--reuse-render-process	Iterates over the sequence of frames to render, and exports Op

Option	Usage
	Tree files for all frames, then starts the renderer (/renderboot process) only once on a sequence of exported Op Tree files. Syntax: `--reuse-render-process` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--reuse-render-process`**



Note: Setting **threads3d** or **threads2d** through Batch mode takes precedence over the **interactiveRenderThreads3D**, and **interactiveRenderThreads2D** settings in Katana's **Edit** > **Preferences** > **application** menu.



Article: How to render an image in multiple tiles in Batch Mode.

Katana Resources

Katana uses the **KATANA_RESOURCES** environment variable to provide a list of paths under which to look for plug-ins and other customizations (such as shelves, tabs, and resolutions). This can also be a list of directories, separated by a colon (if you're on Linux) or a semi-colon (if you're on Windows). The idea is to allow you to build up a list of resource locations. If you're a developer writing plug-ins for Katana, you need to make sure they go in the right place in order for them to be picked up properly.

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

Adding New Paths

Any directories you add to the **KATANA_RESOURCES** path are searched by Katana for plug-in. For example, you could specify:

export KATANA_RESOURCES=/home/tom/dev/katana/Resources:/tools/site/katana/Resources

You should then see the following when Katana starts:

```
> katana
[INFO LicenseCheck]: Interactive License OK
[INFO python.ResourceFiles]: Additional Katana resource paths from $KATANA_
RESOURCES:
[INFO python.ResourceFiles]: /home/tom/dev/katana/Resources
[INFO python.ResourceFiles]: /tools/site/katana/Resources
```

These are searched in addition to those listed in the Defaults section, discussed in more detail below.



Note: The $KATANA_RESOURCES variable behaves as a standard Linux environment variable. Consequently, if you wish to append a directory to this, keeping anything that is already there, you have to take care of when editing this.



Note: If you don't wish to load every plugin stored under your directory, you can set the KATANA_RESOURCES_EXCLUDE_LIBS environment variable to a list of library paths to exclude them from loading at startup.

Important Directories

The paths that you place on **KATANA_RESOURCES** are not actually searched directly. Instead, there is a meaningful set of sub-directories that are used by different parts of the program. For Python modules, the 'types' listed refer to the first value of each tuple set in the module's PluginRegistry list.

- **Args** - the **.args** files for shaders.
- **AssetPlugins** - Python-based AssetPlugin and FileSequencePlugin plug-ins.



Note: As of 3.0v1, Python-based AssetAPI plug-ins have been deprecated. Support for them has been removed when Katana transitioned from Python 2.7 to Python 3.7 as part of moving to VFX Reference Platform CY2020.

- **Engines** - C++ plug-ins that can arbitrarily create and manipulate native USD scene data. Engines can also be placed in the **Libs** sub-folder within **KATANA_RESOURCES**. The Engines are loaded regardless of whether they are placed in the Libs or Engines folders.
- **GafferThree*** - Python-based plug-ins (Script Items) that extend the GafferThree super tool. Custom packages for GafferThree can be created using PackageSuperToolAPI.

- **GenericAssign** - templates for defining new GenericAssign-based nodes (**.xml**).
- **Importomatic*** - Python-based ImportomaticModule plug-ins that form plug-ins for the Importomatic SuperTool.
- **Layouts** - creating a directory called **Layouts** in your **KATANA_RESOURCES** path allows you to load layouts from files named **KatanaLayout2.xml** or other files ending in **.katanalayout.xml**. For example:
  - **<KATANA_RESOURCES>/Layouts/KatanaLayout2.xml**
  - **<KATANA_RESOURCES>/Layouts/production1.katanalayout.xml**

  Layouts saved from Katana's **Layouts** menu are still saved in the **KatanaLayout2.xml** file in the current user's **.katana** resource directory in the OS home directory. For example:
  - **Windows:** C:\Users\<login name>\.katana
  - **Linux:** /home/<login name>/.katana
- **Libs** - any compiled C++ plug-ins for Katana, for any API; for example, C-based asset plug-ins (**.so**).
- **Macros** - nodes saved as macros from the Katana UI (**.macro**).
- **Ops** - C++ plug-ins that can arbitrarily create and manipulate scene data. Ops can also be placed in the Libs sub-folder within **KATANA_RESOURCES**. The Ops are loaded in Katana regardless of whether they are placed in the Libs or Ops folders.
- **Plugins** - sundry Python modules, including types such as GafferProfile, KatanaPlugin, RenderLocationPlugin, ViewerProxyLoader, UVTileFormats.
- **PrimDecorators** - C++ plug-ins that can arbitrarily convert Katana Attributes into USD and are utilised by the KatanaToUsd node. PrimDecorators can also be placed in the Libs sub-folder within **KATANA_RESOURCES**. The PrimDecorators are loaded regardless of whether they are placed in the Libs or PrimDecorators folders.
- **RenderBin** - this directory isn't actually a standard Katana directory, but many render plug-ins use this to store binaries and plug-ins that are loaded by the renders themselves to talk to Katana.
- **Resolutions** - additional resolution files (**.xml**).
- **Shaders** - any additional shaders for a renderer. The only time this directory is considered by Katana itself is to locate shaders for the viewer (**.glsl**).
- **Shelves** - a directory for each shelf, containing Python scripts for each shelf item.
- **Startup** - Python scripts that can be used to configure Katana at startup. The only file that is explicitly run (with execfile) is one called **init.py**. If you wish to run other scripts or import modules, they would need to be called from there.
- **SuperTools*** - Python modules that implement new SuperTools.
- **Tabs*** - Python modules (KatanaPanel) that implement new tabs that can be docked to panes in the UI.
- **UIPlugins** - Python modules that implement UI-specific plug-ins, such as AssetWidgetDelegates and KatanaPlugin.



Note: These plug-ins aren't loaded in **--batch**, **--script**, and **--shell** launch modes.

• **ViewerManipulators** - additional Python manipulators (KatanaManipulator) for the viewer.

Defaults

Katana always looks in the following (internal) places, regardless of what you set **KATANA_RESOURCES** to. **$KATANA_ROOT** is where the Katana installation lives.

```
${KATANA_ROOT}/bin/python/UI4/Resources
${KATANA_ROOT}/plugins/Resources/Core
```

Generally, the search order is by 'standard path behavior'. Namely, Katana looks at the directories, left to right. What you may observe, though, differs a little depending on the type of plugin/directory being loaded.

Compiled Plug-ins

Many of the compiled plug-ins work on the basis that the 'first one loaded' sticks. This is not based on the **.so** name, but instead the name passed to the **REGISTER_PLUGIN**. Repeat registrations with the same name are ignored. The plug-ins in each directory are iterated by readdir so are loaded in 'filesystem order'.

In the case of ViewerModifierPlugins, where the mapping is to a location type (based on an API call), the effective winner for any location is the first named registration that accepts to a particular location type.

But what is the 'first' plug-in? In the case of multiple registrations of the same name, for example, a local build of a central plug-in, both named 'OuModifier', you get the first on the path from left to right. However, when multiple, independently-named registrations handle the same type, for example, 'LightModifier' and 'MyLightModifier', you end up with the first one iterated from the internal plug-in map, which presently ends up being ordered alphabetically.

Python

Python modules are generally sourced from directories (using **__import__**), from left to right. So, the first module that registers a specific name in its PluginRegistry wins. Within any directory, plug-ins are loaded by **os.listdir**, which documents its ordering as arbitrary. However, some code reverses this search order. Any directories listed with an asterisk (*****) above are right-to-left precedence. Additionally, shelves don't work quite as you might expect, as the shelf mechanism searches left-to-right (non-reversed). This means that the right-most files contents win out.

Environment Variables

Environment variables are named variables used to store a value, such as a specific file path, and can be used to influence Katana's behavior. For example, Katana uses the information stored in them to define where to place certain files.

Setting Environment Variables

The section teaches you how to set environment variables, check if a particular environment variable exists, and displays a list of set environment variables.

To Set an Environment Variable

On Linux

1. The procedure for setting an environment variable depends on what your default shell is. To get the name of the shell you are using, launch a shell and enter **echo $SHELL**.
2. Depending on the output of the previous step, do one of the following:
   - If your shell is a csh or tcsh shell, add the following command to the **.cshrc** or **.tcshrc** file in your home directory: **setenv VARIABLE value**. Replace **VARIABLE** with the name of the environment variable and **value** with the value you want to give it, for example, **setenv KATANA_PATH /SharedDisk/Katana**.
   - If your shell is a bash or ksh shell, add the following command to the **.bashrc** or **.kshrc** file in your home directory: **export VARIABLE=value**. Replace **VARIABLE** with the name of the environment variable and **value** with the value you want to give it, for example, **export KATANA_PATH=/SharedDisk/Katana**.

   For a list of the environment variables that Katana understands, see Environment Variables.

On Windows

1. Right-click on **My Computer** and select **Properties**.
2. Go to the **Advanced** tab.
3. Click the **Environment Variables...** button.

   The **Environment Variables** dialog opens.

4. Click the **New...** button either under **User variables** or **System variables**, depending on whether you want to set the variable for the current user or all users. To set environment variables for all users, you need to have administrator privileges.
5. In the **Variable name** field, enter the name of the environment variable you want to set. For a list of the environment variables that Katana understands, see Environment Variables.
6. In the **Variable value** field, enter the value for the variable. The value can be a directory path, for example.
7. Click **OK**.



Note: When editing existing system variables, or adding or deleting either user or system variables, you may need to log off and on again before your changes to environment variables take effect.



Article: How to set environment variables
How to delete or unset environment variables

To Check if an Environment Variable Exists

On Linux

1. Launch a shell.
2. Enter echo **$VARIABLE**. Replace **VARIABLE** wit the name of the environment variable. For example, to check if **KATANA_DISABLE_LIVEGROUP_CACHING** is set, enter **echo $KATANA_DISABLE_LIVEGROUP_CACHING**.

   If the variable is set, its value is displayed in the shell window.

On Windows

1. Select Start > All Programs > Accessories > Command Prompt.
2. In the command window that opens, enter **echo %VARIABLE%**. Replace **VARIABLE** with the name of the environment variable. For example, to check if **KATANA_DISABLE_LIVEGROUP_CACHING** is set, enter **echo %KATANA_DISABLE_LIVEGROUP_CACHING%**.

   If the variable is set, its value is displayed in the command window.



Article: How to get environment variable values

To Display a List of Set Environment Variables

On Linux

1. Launch a shell.
2. Enter **printenv**.
   A list of all environment variables that are set is displayed in the shell window.

On Windows

1. Select **Start** > **All Programs** > **Accessories** > **Command Prompt**.
2. In the command window that opens, enter **set**.
   A list of all the environment variables that are set is displayed in the command window.



Article: How to list environment variables

What is Katana?

Katana was originally designed to solve problems with scalability and flexibility; how to carry out look development and lighting in a way that could deal with potentially unlimited amounts of scene data. It also needed to be flexible enough to deal with the requirements of modern CG Feature and VFX production for customized workflows, with the capability to edit or override anything.



Video: This video gives you an overview of what Katana is.

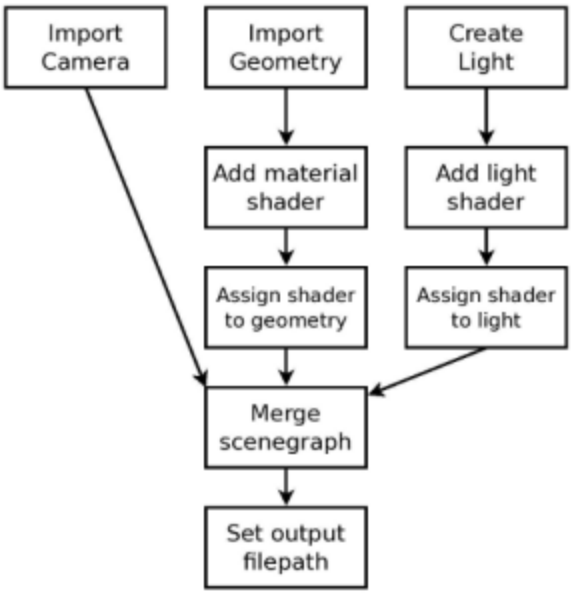
Katana leverages renderers' support for recursive procedurals, where arbitrary scene data can be created on demand. The Katana approach is to have a single procedural that is powerful enough to handle arbitrary generation and filtering. Essentially, this is a procedural given a custom program in the form of a tree-based description of filters. At render time, Katana's libraries are called from within this procedural to calculate scene data as the renderer demands.

What Can Katana Do?

Katana allows you to define what to render by using filters that can create and modify 3D scene data. A node-based interface allows users to define which filters to use, and interactively inspect their results.

Using filters you can arbitrarily create and modify scene data. You can, for example:

- Bring 3D scene data in from disk, such as from an Alembic geometry cache or camera animation data.
- Create a new instance of a material, such as a 3Delight shader.
- Create cameras and lights.
- Manipulate transforms on cameras, lights and other objects.
- Use rule based expressions to set what materials are assigned to which objects.
- Isolate parts of the scene for different render passes.
- Merge scene components from a number of partial scenes.
- Specify which AOV's you want to use for multiple passes in a single render..
- Use Python scripting to specify arbitrary manipulation of attributes at any location in the scene hierarchy.



The scene data to be delivered to the renderer is described by a tree of filters, and the filters are evaluated on demand in an iterative manner. Katana is designed to work well with renderers that are capable of deferred recursive procedurals. Using recursive procedurals, the tree of filters is handed directly to the renderer, with scene data calculated on demand, as the renderer requests it (lazy-evaluation). This is typically done by a procedural inside the renderer that uses Katana libraries, during render, to generate scene data from the filter tree.

Katana can also be used with renders that don't support procedurals or deferred evaluation, by running a process that evaluates the scene graph and writes out a scene description file for the renderer. This approach is without the benefits of deferred evaluation at render time, and the scene description file may be very large.

Note: Since Katana's filters deliver per-frame scene data in an iterable form, Katana can also be used to provide 3D scene data for processes other than renderers.

At its core, Katana is a system for the arbitrary creation, filtering, and processing of 3D scene data, with a user interface primarily designed for the needs of look development and lighting. Katana is also designed for the needs of power users, who want to create custom pipelines and manipulate 3D scene data in advanced ways.

Scene Graph Iterators

The key to the way Katana executes, filters, and delivers scene data on demand, is that scene data is only ever accessed through iterators. These iterators allow a calling process (such as a renderer) to walk the scene

graph and examine any part of the data on request. Since that data can be generated as needed, a large scene graph state doesn't have to be held in memory.

In computer science terms, it is the responsibility of the calling process to maintain its own state. Katana provides a functional representation of how the scene graph should be generated, that can be statelessly lazily-evaluated.

At any location in the scene hierarchy Katana provides an iterator that can be asked:

- What named attributes there are at that location?
- What are the values for any named attribute (values are considered to be vectors of time sampled data)?
- What are the child and sibling locations (if any)?

Katana in Look Development and Lighting

Katana's scene generation and filtering are presented as a primary artist facing tool for look development and lighting by having filter functions that allow you to perform all of the classic operations carried out in look development and lighting. Primarily:

- Creating instances of shaders, or materials, out of networks of components
- Assigning shaders to objects
- Creating lights
- Moving lights
- Changing visibility flags on objects
- Defining different render passes

Katana's node-based interface provides a natural way to create recipes of which filters to use. Higher-level operations that may require a number of atomic level filters working together can be wrapped up in a single node so that the final user doesn't have to be concerned with every individual fine-grain operation. Multiple nodes can also be packaged together into single, higher-level compound nodes.

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.

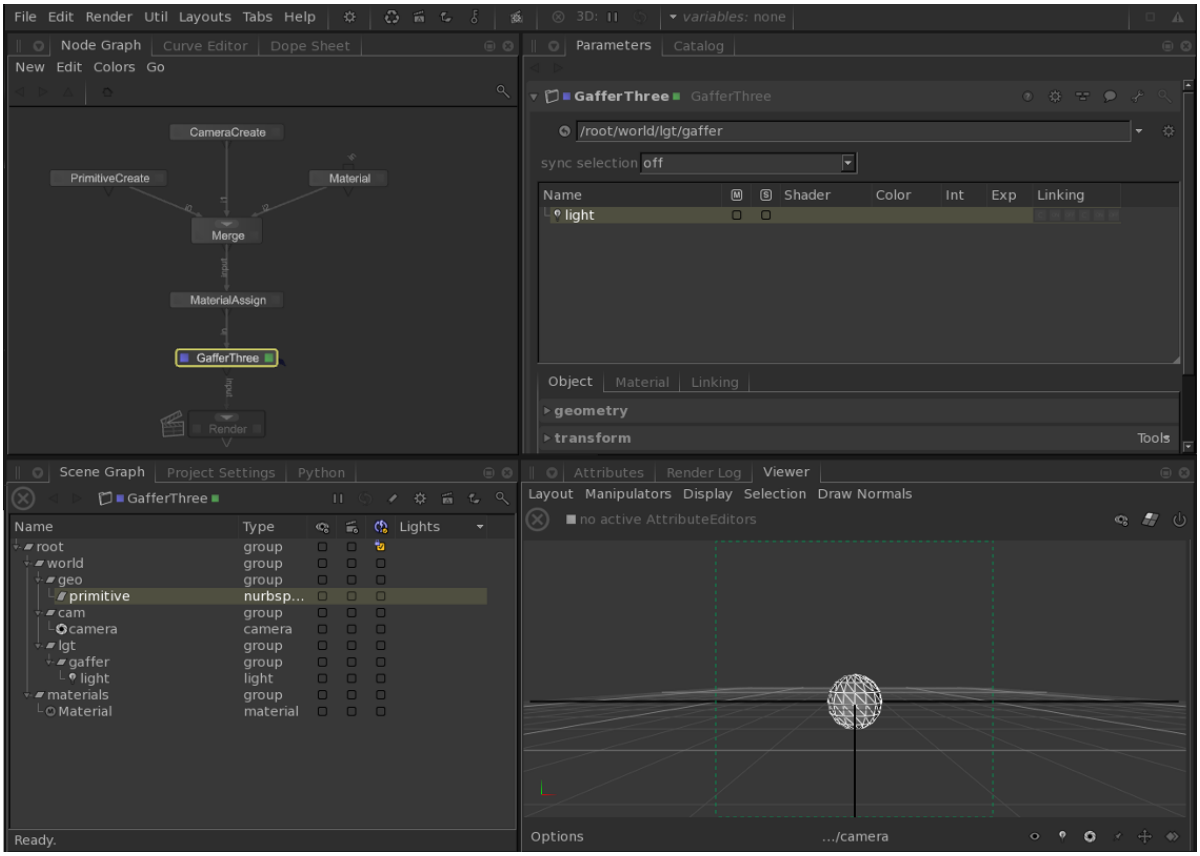
To view the scene generated up to any node within a recipe, you use the **Scene Graph** tab. The scene graph's hierarchical structure is made up of locations that can be referenced by their path, such as **/root**. Each location has a number of attributes that represent the data at that location. You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

These key concepts are explained in greater depth in the Quick Start Guide.

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.



The User Interface

Katana allows you to create recipes for filters, using a familiar node-based user interface (UI). In the UI, you can also interactively examine the scene at any point in the node tree, using the same filters that the renderer runs at render time (but executed in the interface).

When running through the UI, filters are only run on the currently exposed locations in the scene graph hierarchy. This means you can inspect the results of filters on a controlled sub-set of the scene.

The way you can view the scene generated at any node is similar to the way users of 2D node-based compositing packages can view composited frames at any node. If you are accustomed to conventional 3D packages that have a single 3D scene state, it may be a surprise that there is essentially a different 3D scene viewable at each node. Instead of the scene graph being expanded as rays hit bounding boxes, it is iterated as you open up the scene graph hierarchy in the UI. Complexity is controlled by only executing filters on locations in the scene graph that you have expanded.

A scene does not need to be entirely loaded in order to be lit. In Katana, you create recipes that allow scene data to be generated, rather than directly authoring the scene data itself. It is only the renderer that needs the ability to see all of the scene data, and then only when it needs it. Katana provides access to any part of the scene data if you need to work on it. You can set an override deep in the hierarchy, or examine what attribute values are set when the filters run, but you can work with just a sub-set of the whole scene data open at a time. This is key to how Katana deals with scenes of potentially unlimited complexity.



Note: As Katana uses procedurally defined iterators, it's possible to define an infinitely sized scene graph, such as a scene graph defining a fractal structure. An infinite scene graph can never be fully expanded, but you can still work with it in Katana, opening it to different depths, and using rule-based nodes to set up edits and overrides.



Note: Katana 2.x uses an application-wide Qt style sheet to apply font preferences to Qt widgets. Custom widgets that use font metrics before widgets are shown need to be modified to add **QWidget.ensurePolished()** calls before working with **QtGui.QFontMetrics** instances.

Glossary of Katana Terms

This glossary provides short descriptions of the most important terms used throughout the Katana application and documentation.

Knowledge of these core terms help you to understand the way Katana works and processes data more clearly, and enable you to make the most of the Katana documentation.

Katana Core Terms

Nodes	**Nodes** are the units used in the Katana interface to build the **Recipe** for a Katana project. **Nodes** feature Parameters that can be used to control their behavior. **Nodes** can be created and connected in Katana's Node Graph tab in the UI, and can also be modified through Python scripting using NodegraphAPI. Katana ships with many built-in types of nodes, but custom node types can also be created through Python scripting. There are two major groups of node types shipped with Katana: • **3D nodes** that produce scene graph that can be inspected in Katana's Scene Graph tab. • **2D nodes** that produce images that can be viewed in Katana's Monitor tab. **Nodes** and their Parameters effectively represent and control corresponding Ops that form Op graphs that are processed by Katana's geometry library to generate the scene data that can be viewed and inspected in Katana's Scene Graph and Attributes tabs.  **Note:** For more information about working with nodes in the Katana UI, please see, Editing the Node Graph. For working with nodes through Python Scripting, please see the relevant Working With Nodes section of the Katana developer guide.
Node Graph	**Node Graphs** in Katana are Recipes of connected nodes that are part of a Katana project. The nodes in **node graphs** can be created and connected in Katana's **Node Graph** tab in the UI, and can also be modified through Python scripting using functionality from the NodegraphAPI Python package.

Parameters	**Parameters** are a part of nodes, and typically control their respective node's behavior. **Parameters** of nodes can be edited in Katana's Parameters tab in the UI, by setting the edit flag on a node in the Node Graph tab, and can also be edited through Python scripting using parts of the NodegraphAPI. Values of parameters can be either constant, determined by Python expressions, or driven by animation curves.
Recipe	**Recipes** in Katana are node graphs of connected **nodes** that are part of a Katana project. **Recipes** typically represent the steps taken or operations performed to create 3D scene data in a scene graph, or the image manipulations performed to create 2D images that can be viewed in Katana's Monitor tab and written out to file.  **Note:** For more information about **Recipes** in Katana, see Creating a Katana Project.
Project	A Katana **Project** is the sum of all the nodes and their parameters that form the recipes that are expressed in the project's node graphs. **Projects** are saved in Katana project files with the **.katana** file extension.  **Note:** For more information on working with projects, please see Creating a Project.
Ops	**Ops** are the building blocks of operations that create and manipulate 3D scene data in Katana, and produce the scene graphs that can be inspected at any point in a Katana node graph by setting the view flag on a particular node. **Ops** are instances of Op Types, which are plug-ins written in C++ that use a particular Katana API to define their inner workings: the Op API. Some functions available for C++ Ops are documented in the Katana Developer Guide. Similar to the various node types, Katana ships with many built-in types of **Ops**, but custom Op types can also be created through C++ programming and using the Op API. When the view flag on a node is set, the node is queried for its corresponding **Ops**. The behavior of a node in terms of the creation or modification of 3D scene data can be defined by a single **Op**, but can also be defined by a number of **Ops** arranged in an Op Chain or Op Graph.
Op Arguments	**Op Arguments** control the behavior of **Ops** that define the effect of **nodes** in a Katana recipe. They roughly correspond to **parameters** on 3D nodes. When changing the parameter of a node, corresponding Op Arguments are updated. If the node or any

	node downstream is being viewed, the scene is recooked.
Cooking	**Cooking** is the act of executing the **Ops** that correspond to nodes in the Katana recipe, in order to create scene graph locations and their attributes, which can then be viewed and inspected in the **Scene Graph** and **Attributes** tabs. When setting the view flag on a node in the node graph, the Ops that correspond to that node and all of the nodes above it are executed/evaluated/cooked to produce the scene graph at that point in the node graph. In technical terms, the **cook()** function of each corresponding Op type plug-in, is being called to create or modify locations in the resulting scene graph.
Filters	**Filters** are the old equivalent in Katana 1.X releases, of Ops in Katana 2.X releases and above. They represent the building blocks of operations that create and manipulate 3D scene data in Katana 1.X releases.
Lazy Evaluation	One of the key aspects of Katana's processing paradigm, is that operations are only evaluated when their results are needed. For example, the Ops that correspond to a particular node are only cooked when the node itself or a node downstream of it is being viewed, meaning it has its view flag set. In the context of the **Scene Graph** tab, data for scene graph locations is only produced when the scene graph hierarchy is expanded to reveal them in the tree view widget. When working with Katana's APIs, **lazy evaluation** can have an effect on the results of certain function calls. **Note:** For an example of this, see the Knowledge Base Article How to Query Attributes of Scene Graph Locations via Python using a Geolib3 Client. **Lazy evaluation** also applies to aspects of Katana's UI, where a mechanism named freezing and thawing ensures that the UI is only updated when necessary in response to user interactions.
Graph State	Katana maintains a **Graph State** data structure when traversing up the node graph. It contains information such as the current frame and the shutter timings, and is passed to **Ops** that are represented by nodes when cooking the scene graph. Nodes can read from and write to the **Graph State** as part of identifying their inputs. For example, a **TimeOffset** node reads the current time and increments or decrements it by some amount, as controlled by its inputFrame parameter. The modified **Graph State** is then passed to the node above for cooking its Ops. It is important to realize

	that the **Graph State** information flows up the node graph, unlike scene data, which flows down the graph. Some Python functions to work with **Graph State** are documented in the Katana Developer Guide.
Graph State Variables	**Graph State Variables** (sometimes abbreviated as **GSV**) essentially allow users 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, where groups of nodes and entire node graph branches can be enabled and disabled with ease.  **Note:** For more information, see Graph State Variables. Project-level **GSV** are known as **Global Graph State Variables**, and node-level **GSV** are known as **Local Graph State Variables**. The following types of nodes are available for working with and/or modifying local **GSV**: • VariableSet • VariableSwitch • VariableEnabledGroup • VariableDelete
GenericAssign	**GenericAssign** is an advanced and powerful concept in Katana, in which parameters of nodes are associated with specific attributes on locations in the scene graph. Such parameters effectively control the values of their corresponding attributes. Their widgets in **Parameters** tabs are capable of showing values of attributes from the incoming scene, allowing users to inspect and modify those attribute values. An example of a node type that uses **GenericAssign**-based parameters is the **RenderSettings** node type. The parameters of **RenderSettings** nodes correspond to attributes in the **renderSettings** group attribute on the **/root** location in the scene graph. When setting a value of a parameter of a **RenderSettings** node, the corresponding attribute in the **renderSettings** group is set. When connecting a **RenderSettings** node to an incoming node graph, the widgets of parameters of the node show the values of the attributes they correspond to. State badges that are part of the parameters' widgets show the value states of the respective parameters, indicating whether the corresponding attributes are set to a specific value by nodes upstream of the node that is being edited (incoming value), or

	by the node itself (local value), or whether the attributes are not set to a specific value, in which case they use a default value instead.
Scene Graph	3D nodes that are part of Katana recipes produce a hierarchical set of data, called the **Scene Graph**, which can be interactively inspected in Katana's **Scene Graph** and **Attributes** tabs in the UI, and can be presented to a renderer or any output process. Examples of data that can be held in the scene graph can include: • Geometry • Particle data • Lights • Instances of shaders • Global option settings for renderers. **Note:** For more information on the **Scene Graph**, see Using the Scene Graph.
Locations	**Locations** are the units that make up the **Scene Graph** hierarchy. Many other 3D applications refer to these as nodes, but in Katana they are referred to as locations to avoid confusion with the nodes used in the Node Graph. **Locations** can uniquely be identified using their name and the names of all of their ancestor locations in the scene graph, which form a scene graph location path, for example: /root/world/geo/pony **Note:** For more examples of how to work with **locations** in the Scene Graph, see Using the Scene Graph and Manipulating the Scene Graph.
Attributes	**Attributes** are containers for data held on locations in the scene graph. Examples of data stored in attributes are: • 3D transforms such as 4x4 matrices • Vertex positions of geometry, • Value settings for an instance of a shader. **Attributes** of a selected scene graph location can be inspected interactively in Katana's **Attributes** tab, but not edited, as their values are determined by nodes and parameters of the Katana project.

	 **Note:** For examples of common attributes that locations can have, see Attribute Conventions in the Katana Developer Guide. For more detailed information on creating, manipulating or deleting attributes, see Working with Attributes.
Attribute Types	There are different types of attributes for different basic types of data: • Integer numbers • Floating-point numbers • Double-precision numbers • Strings In addition to these types of data attributes, attributes can be grouped in hierarchies using group attributes. A special type of attribute, the **null** attribute, is used for specific cases, such as to declare a certain attribute as not set, so that a default value for the attribute is used instead.

Katana UI Terms

Value Policies	**Value Policies** in Katana provide data for display in widgets in Katana's UI. **Value policies** provide a layer in between underlying data sources, such as parameters of nodes in the node graph document, and UI widgets in tabs like the **Parameters** tab. There are different types of **value policies**, tailored to specific data sources and specific use cases. The Python base class for **value policies** is **QT4FormWidgets.AbstractValuePolicy**. **Value policies** take care of translating from events in the underlying data sources to Qt widget events, for example, to repaint widgets after a parameter's value has been changed using a NodegraphAPI call.
Parameter Policies	**Parameter Policies** in Katana are value policies that provide a layer in between parameters of nodes in the node graph document and widgets in the **Parameters** tab. Those widgets show values of parameters, and can be used to edit those parameter values.

	Parameter policies are most relevant when developing parameter UIs for custom types of nodes, for example **SuperTools**, using Python scripting APIs.  **Note:** For examples of how this can be used in SuperTools, please see the respective **Editor.py** files of the example **SuperTools** that ship with Katana under: **$KATANA_ ROOT/plugins/Src/Resources/Examples/SuperTools** The Python base class for parameter policies is **UI4.FormMaster.BaseParameterPolicy**. It is derived from the **AbstractValuePolicy** class. A parameter policy is typically created for a specific parameter of a specific node by passing a **NodegraphAPI.Parameter** instance that represents the respective parameter to the **UI4.FormMaster.CreateParameterPolicy()** function. This returns an instance of a class that is derived from the **BaseParameterPolicy** class.
Attribute Policies	**Attribute Policies** in Katana provide a layer in between attributes of locations in the scene graph and widgets in the **Attributes** tab that show values of those attributes. **Attribute policies** are created internally by Katana to provide attribute data for display in the **Attributes** tab when locations in the **Scene Graph** tab are selected. There's rarely a need to create attribute policies manually. The Python base class for Geolib3-based attribute policies is **UI4.FormMaster.FnAttributePolicy.AttributePolicy**. It is derived from the **AbstractValuePolicy** class.
GenericAssignParameterPolicy	**GenericAssign Parameter Policies** (**GAPP**) in Katana are parameter policies that provide a layer in between **GenericAssign**-powered parameters of nodes in the node graph document and widgets in the **Parameters** tab. They can be seen as a hybrid between parameter policies and attribute policies: • **GenericAssign aspect**: **GAPP**s receive the results of cooking

	the scene graph that is produced by nodes upstream of the respective **GenericAssign**-powered node by way of a built-in Geolib3 Client that receives events from the Geolib3 Runtime. This is similar to how attribute policies provide data from attributes of scene graph locations for display in the **Attributes** tab. • **Parameter policy aspect**: **GAPP**s provide cooked attribute data for use in parameter widgets in the **Parameters** tab. This is similar to how parameter policies provide data from parameters of nodes for display in the **Parameters** tab.
Freezing and Thawing	**Freezing** means that Katana-specific events that are normally processed when underlying data changes are temporarily ignored. It applies to value policies, and to tabs in Katana application windows. This ensures that they're not needlessly updated when changes are made to the node graph document or when scene graph location data is cooked, for example in the case that widgets or tabs are not actually visible to the user. **Thawing** of **value policies** or tabs is the reverse of freezing: after the processing of Katana-specific events has temporarily been suspended, or has never been started before, thawing means that processing of such events and updating UI components as a result is resumed or started. **Freezing** and **thawing** is implemented by registering and unregistering handlers for specific Katana event types, depending on whether the respective value policies or tabs are frozen.  **Note:** For information on registering callbacks and event handlers, refer to Callbacks and Events in the developer guide. Typically, when the user switches from one tab to the next in a pane inside of a Katana window, the previously visible tab is frozen, and the newly visible tab is thawed. Thus, no widgets in the now hidden tab are updated in response to Katana events, but widgets in the newly visible tab are. When working with parameter policies in the context

	of parameter UIs for custom types of nodes, for example **SuperTools**, it is important to note that Python callback functions need to be added to such a **value policy** in order to be notified when the underlying value of the policy changes. If no such callbacks are added to a value policy, the value policy is considered frozen.  **Note:** For more information, see the descriptions of **GenericAssign** and **Lazy Evaluation** in the Katana Core Terms section of the Glossary, and GenericAssignParameterPolicy in the UI Terms section.

ScenegraphManager	The **ScenegraphManager** Python module is part of the **Nodes3DAPI** Python package. It maintains a single instance of a **Scenegraph** class that is responsible for tracking a number of Working Sets that maintain the open, closed, selection, and pinning states of locations in Katana's scene graph.
	
	For more information about pinning, see Changing What is Shown in the Viewer.
	
	The **Scenegraph** instance can be retrieved by calling **ScenegraphManager.getActiveScenegraph()**. The instance can then be used, for example, to access the list of paths of scene graph locations that are currently selected:
	`sg = ScenegraphManager.getActiveScenegraph()`
	`print(sg.getSelectedLocations())`
	
	The **Scenegraph** class also maintains a history of selected scene graph locations, using an internal **SelectionHistory** class, for the purpose of allowing users to step through the history using the **History Forward** and **History Backward** commands in the **Viewer** tab.

Katana Rendering Terms

Preview Render	A **Preview Render** is a type of **Interactive Render**, meaning a render launched from a

	Katana UI session, in which the rendered image and a progress bar are displayed in Katana's **Monitor** tab. In a **Preview Render**, the renderer process quits when the rendered image is completed. This is different to a Live Render, in which the renderer process is kept alive. **Preview Rendering** has historically been called interactive rendering in early versions of Katana. **Note:** For more information about **Preview Rendering**, see Performing a Render.
Live Render	A **Live Render** is a type of **Interactive Render**, meaning a render launched from a Katana UI session, in which the renderer process is kept alive while the rendered image is displayed in Katana's **Monitor** tab. When making changes to parameters of nodes in the Katana project, the renderer is notified of these changes, and the rendered image updated in the **Monitor** tab. It's possible to limit for which scene graph locations updates are sent to the renderer during a **Live Render** session, by using the **Live Render Updates** column in Katana's **Scene Graph** tab. This is typically used for projects that produce very large scene graphs. **Live Rendering** has historically been called re-rendering in early versions of Katana. **Note:** For more information about **Live Rendering**, see Performing a Render.
Disk Render	A **Disk Render** is a type of render in which the rendered image is written to a file on disk, and then loaded into the **Monitor** tab when the render has finished. The progress bars in the **Monitor** tab are not updated while a **Disk Render** is in progress. **Disk Rendering** has historically been called hot-rendering in early versions of Katana. While the Preview and Live Render options are available from any node's context menu, a **Disk Render** can only be triggered from a **Render** node. **Note:** For more information about **Disk Rendering**, see Render Types.

Interactive Render	An **Interactive Render** is a render launched from a Katana UI session. There are two types of interactive renders available in Katana: • Preview Render • Live Render
Interactive Render Filters	**Interactive Render Filters** (commonly abbreviated as **IRF**) allow users to set up common recipe changes for **interactive renders**, meaning Preview Renders and Live Renders, without having to add nodes to effect such changes at various points in a project's recipe. An **IRF** can consist of more than one change to the recipe, and it is the equivalent of appending nodes to the end of the node from which an interactive render is started. **IRF**s are defined in **InteractiveRenderFilters** nodes and can be selectively activated and deactivated in the **Interactive Render Filters** popup. This popup is accessible by clicking the **Interactive Render Filters** button  at the top of the Katana interface. **Note:** **Interactive Render Filters** are ignored for Disk Renders. An example use for **IRF**s is to set them up to reduce the render resolution for interactive renders without affecting Disk Renders, thus making debugging of such renders much quicker. Other examples of changes that can be set up by **IRF**s might include anti-aliasing settings, shading rate changes, or the number of light bounces. **Note:** For more information about setting up **Interactive Render Filters**, see Setting up Interactive Render Filters. You can also refer to the Knowledge Base Article Increasing preview efficiency with Interactive Render Filters.
Render Dependency	When starting a render from a **Render** node, other render passes that the render may depend on can be rendered to disk automatically by rendering with dependencies. Historically, this feature was used to produce shadow maps that are then used for a main render pass. **Note:** For more information, see Render Dependencies.

User Interface

This section walks you through the main components of the Katana UI.

The Default Workspace

An illustrated overview of the Katana workspace.

The Default Tabs

Tabs are themed panels that present Katana functionality.

Menu Bar Components

A list of the functions available in the menu bar. Plus details of the Message and Notification Centers.

Customizing Your Workspace

A brief introduction to changing the layout.

Adjusting Layouts

How to modify the layout of the tabs.

Saving, Loading, and Deleting Layouts

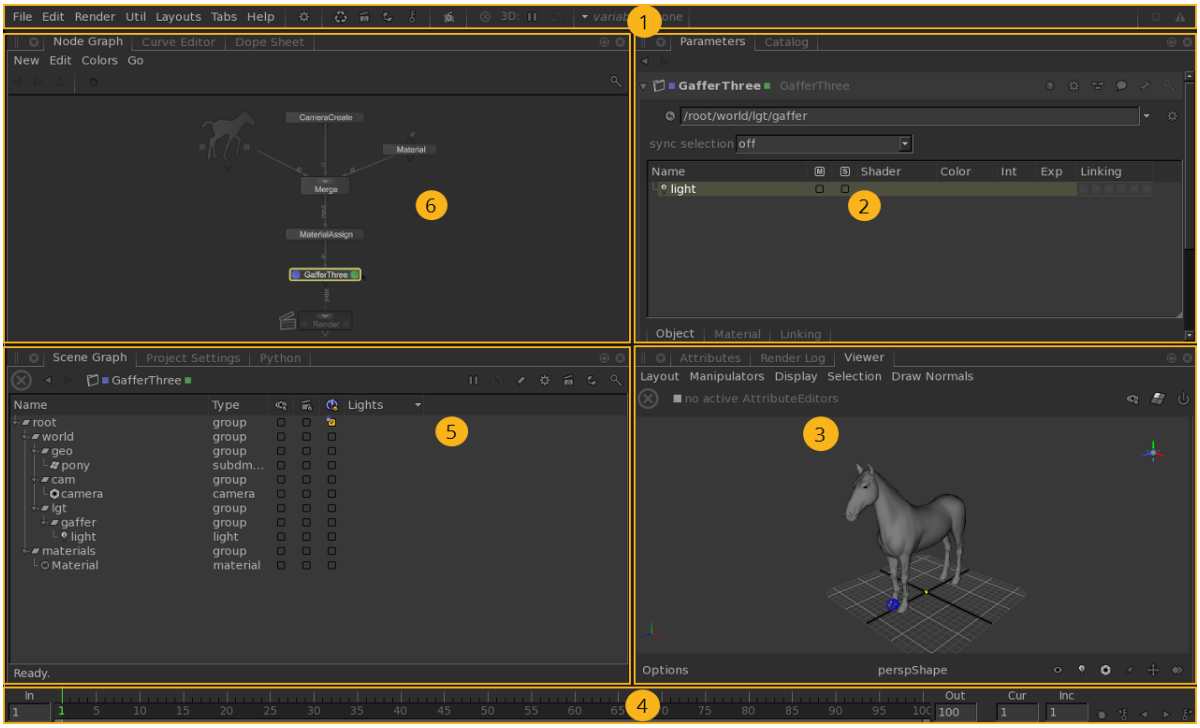
Katana lets you save your preferred layout.

Managing Keyboard Shortcuts

About the shortcuts.xml file.

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-right pane, containing the **Parameters** and **Catalog** tabs.
3. The bottom-right pane, containing the **Attributes**, **Render Log**, and **Viewer** tabs.
4. The **Timeline**. The **Timeline** is explained in greater depth in Using the Timeline.
5. The bottom-left pane, containing the **Scene Graph**, **Project Settings**, and **Python** tabs.

   For more on the contents of the various tabs, see the The Default Tabs below.
6. The top-left pane, containing the **Node Graph**, **Monitor**, **Curve Editor**, and **Dope Sheet** tabs.

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).

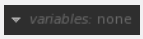
Tab	Function
Monitor	This is where you view the results of your renders and composites.
Curve Editor	Lets you edit animation keys as curves.
Dope Sheet	Lets you edit animation keys as a spreadsheet of keys and ranges.
Scene Graph	This is where you view the scene data, generated at the current view node 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.

Menu	Functions
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.
	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**).

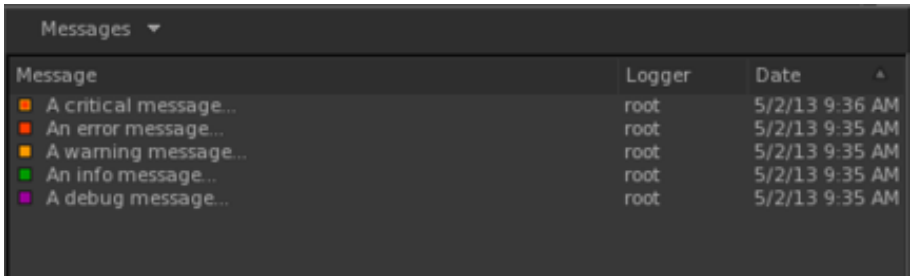
Menu	Functions
	When live rendering is set to **Manual**, this button triggers an update. This button is not clickable for either **Pen-up** or **Continuous** modes.
	Displays any graph state variables that have been set in the **Project Settings** tab. If no variables have been set, the icon says **variables: 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 Graph State Variables.
	When enabled, displays the Message Center and any messages contained therein.
	When enabled, displays the Notification Center and any notifications contained therein.

The Message Center

Katana uses the standard Python logging module and, by default, logs messages of types info, warning, error, and critical. There is also a debug message that can be enabled from within the Message Center. Katana records messages on activities such as loading scenes, and converting scripts between different render versions. The message button is on the right-hand side of the menu bar, and uses the following colors to categorize the messages:

- Critical Messages - marked orange .
- Error Messages - marked red .
- Warning Messages - marked yellow .
- Information Messages - marked green .
- Debug Messages - marked purple .

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 any of those listed above, or unfilled). The Message Center shows a truncated summary of each message. If you select the message and copy it, you also copy the full text, which you can then paste into a text editor.



Clicking **Messages** within the Message Center opens the dropdown menu where you can enable or disable the display of specific message categories, copy selected messages, or delete selected messages.

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.

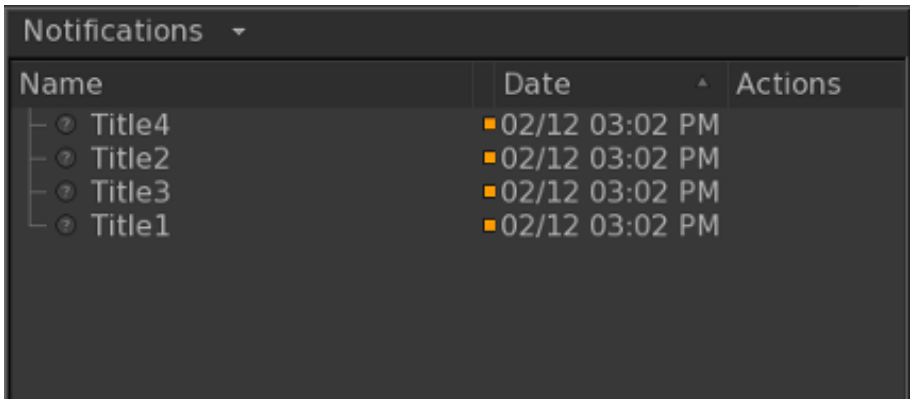
For more information on message logging, using either the C++ or Python methods, see Message Logging.

The Notification Center

Katana uses the standard Python logging module to record user notifications. These differ from the messages in the Message Center, as they are not designed to be related to error messaging, so much as internal messages you want other users to be aware of. Below is an example of how you can use the NotificationManager class to record or display notifications in the Notification Center:

```
for i in range(4):
    notificationRecord = UI4.Util.NotificationManager.NotificationRecord
('Title%d' % (1 + i), 'Text %d' % (1 + i))
    UI4.Util.NotificationManager.AddRecord(notificationRecord)
```

If triggering the NotificationManager causes new notifications to be logged, the Notification Center icon lights up . Click it to toggle the Notification Center window. Any notifications that haven't been deleted from the Notification Center are displayed in the window, along with the date of the notification, whether any action is necessary, and additional comments, if applicable.



Clicking **Notifications** within the Notification Center opens the dropdown menu where you can ignore, unignore, or delete, specific notifications. You can also right-click on any given notification for the same options as those in the **Notifications** dropdown. Clicking on the help icon, opens the full notification text in a separate window.

Customizing Your Workspace

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.

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.

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). Additionally, you can toggle between viewing the main Katana window in fullscreen mode or standard mode by selecting **Layouts** > **View Fullscreen**. Below are more useful layout changes that may help you customize Katana to your own preference.

Panes

You can resize individual panes, by hovering 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.

If you want to maximize a pane so that it expands to the size of the window:

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

Alternatively, you can return to the regular interface, by clicking or pressing **Spacebar**.

Note: If you have any tabs dock widgets, these remain in place when you maximize a pane.

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

You can move an existing pane to a new location in the interface by hovering over the move pane icon in the top-left corner of the pane until the cursor changes to the move icon, then clicking and dragging 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.

If you want to add a floating pane to the interface, click **Tabs** > **[tab name]**.

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

Adding Tabs

You can add a tab to a specific pane by clicking  in the top-left corner of the pane and selecting the tab you want to add. If you then want to move that tab, or another existing tab, to a new location in the interface, click and drag it 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.

Floating Windows

If you want to turn a tab into a floating window, right-click on the name of the tab and select **Detach tab**. Alternatively, if you want 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.

Docking Tabs

To dock a tab within Katana's main window, right-click on the name of the tab and select **Move Tab To**, then select from the following options:

- Left Dock
- Right Dock
- Top Dock
- Bottom Dock

Tabs in dock widgets can be dragged to one of the other dock widget areas, and can also be turned into floating panes by clicking the **Detach Tab** button  in the title bar of the dock widget, by double-clicking its title bar, or by dragging a docked tab away from the dock widget areas of the main window.

Docked tabs are saved and restored as part of Katana layout XML files. A **Save # Dock Widgets** checkbox has been added to the **Save Current Layout** dialog that opens when choosing the **Layouts** > **Save Current Layout** menu command. By default, the checkbox is turned on. The checkbox can be turned off to not save docked tabs as part of the layout.

Timelines

There are a few visibility options for timelines in tabs. You can show or hide a timeline at the bottom of the main Katana window by selecting **Layouts** > **Show Main Timeline**, or you can show or hide a timeline at the bottom of any tab by right-clicking on the tab name and selecting **Show Timeline**.

Editing Node Type Parameters

Clicking the tabs  icon, and selecting **Node** opens a node type panel. This allows you to edit the parameters of nodes of the specified type, for example the parameters of several GafferThree nodes. By default, the options under  > **Node** are **GafferThree**, **LookFileManager**, and **MaterialStack**. If you want to add other node options to the **Node** dropdown, set the **KATANA_NODETYPETAB_NODETYPES** environment variable with the name of the node or nodes, separated by commas. For example, **KATANA_NODETYPETAB_NODETYPES=CameraCreate,PrimitiveCreate**.



Note: If you specify nodes with the **KATANA_NODETYPETAB_NODETYPES** environment variable, it overwrites the default Katana nodes specified under  > **Node**. If you want to retain these nodes, in addition to those you specify, they need to be added to the environment variable list.

You can also access these options from **Tabs** > **Node**.

To remove individual tabs, make sure you are viewing the tab you want to remove and click on the close tab  icon in the top-right corner of the pane, or right-click on the name of the tab and select **Close tab**.

Saving, Loading, and Deleting Layouts

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. If your layout includes any dock widgets and you want those to be saved with the layout, check **Save # Dock Widgets** (where # corresponds to the current number of dock widgets).
5. 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**.

Managing Keyboard Shortcuts

The `$HOME/.katana/shortcuts.xml` configuration file can be used to override the default keyboard shortcuts of actions and key events that are registered with Katana's new Keyboard Shortcut Manager.

Example of a shortcuts.xml File

Below is an example of a `shortcuts.xml` file:

```
<shortcuts>
  <shortcut id="430f81d33d338680a0c64ae9ea311cd7"
            name="SceneGraphView.ExpandBranchesToAssembly"
            shortcut="A"></shortcut>
</shortcuts>
```

The ID of a keyboard shortcut element is assigned by the developer 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 changes, 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 in future Katana releases.

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 view the currently assigned keyboard shortcuts of actions and key events, for which custom keyboard shortcuts can be assigned, in the **Keyboard Shortcuts** tab. 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 selecting **Copy as XML** from the context menu. Pasting such an XML representation into the `shortcuts.xml` file allows you to override the custom keyboard shortcut assigned for the respective action or key event.

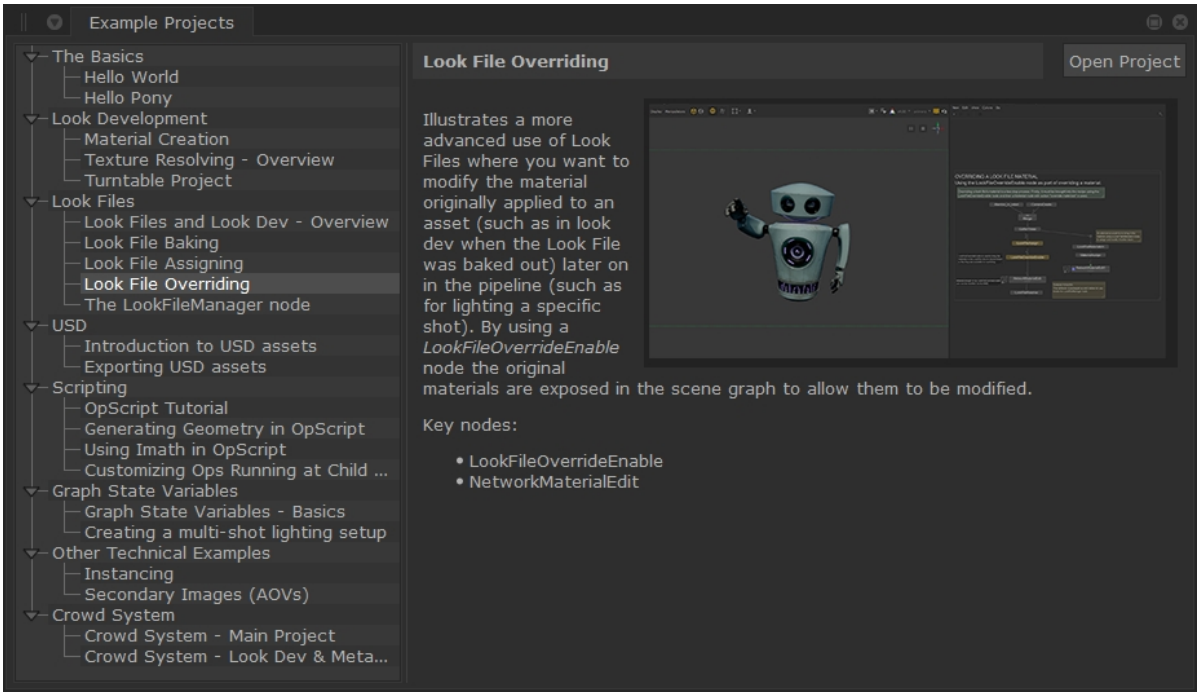
In future releases of Katana, more and more of Katana's menu commands and other actions and key events are adopted to using the new Keyboard Shortcut Manager, so that they can be customized as well.

Getting Help

In the scope of this user guide, it's not possible to go into detail with Python and all the scripts available. However, there are several sources of more information that you may find useful if you need help using Python.

Example Projects

Katana ships with a number of example projects covering a wide range of topics from **The Basics** and **Look Development** to **Scripting** and **Graph State Variables**. To load a project, navigate to **Help** > **Example Projects** to display a list of available scripts.



You can double-click a project in the list or select the project you want to load and click **Open Project**.

API Reference

In the API References, you may find some of the things you need in terms of Python examples and ways of using Python in Katana. You can navigate to these references by clicking **Help** > **API Reference** and selecting the required API from the dropdown menu, or by clicking **Help** > **Developer Guide**, which leads you to a comprehensive page that links you to scripting resources, API references (including plug-in APIs), and legacy documentation.

Viewing More Examples

Python

Only a few samples of Python are described in this section but there are scripts also available in the following locations:

• On Windows :

drive letter:\Program Files\Katana7.5v5\plugins\Src

• On Linux:

/usr/local/Katana7.5v5/plugins/Src/Resources

To view an example, select one of the **.py** files from the **Examples** folder and open it in any text editor or, to see what Python files Katana uses as part of the application, view the **.py** files in the **Core** folder.

Lua

Only a few samples of Lua are described in this section but there are also scripts available in the The Op API section and within the application under **Help** > **Example Projects**. Any of the OpScript-related projects under the **Scripting** section contain Lua examples.

C++

Only a few samples of C++ are described in this section, but there are also scripts available in the following locations:

On Windows :

drive letter:\Program Files\Katana7.5v5\plugins\Src

• On Linux:

/usr/local/Katana7.5v5/plugins/Src

To view an example, select one of the **.cpp** files and view them in any text editor.

Using the Help Function

If you are working in the **Python** tab, one of the quickest ways of getting help on specific things, is to call the help function with the object you're interested in. For example, the following statement gives you a description of what the **setattr** function does:

help(setattr)

This generates the help text for the specified function in the output pane.

If you're unsure how the function should be written or completed, begin typing and then press **Tab**. A list of possible matches appears in the output pane.

Creating a Project

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 that creates a **Katana look file** (**.klf**) for a piece of geometry and another recipe that 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.

If you'd like a quick start guide to take you through creating nodes to rendering, have a look at the Quick Start Guide page in this section.

Creating, Saving, and Loading a New 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 Project

To save your current Katana project:

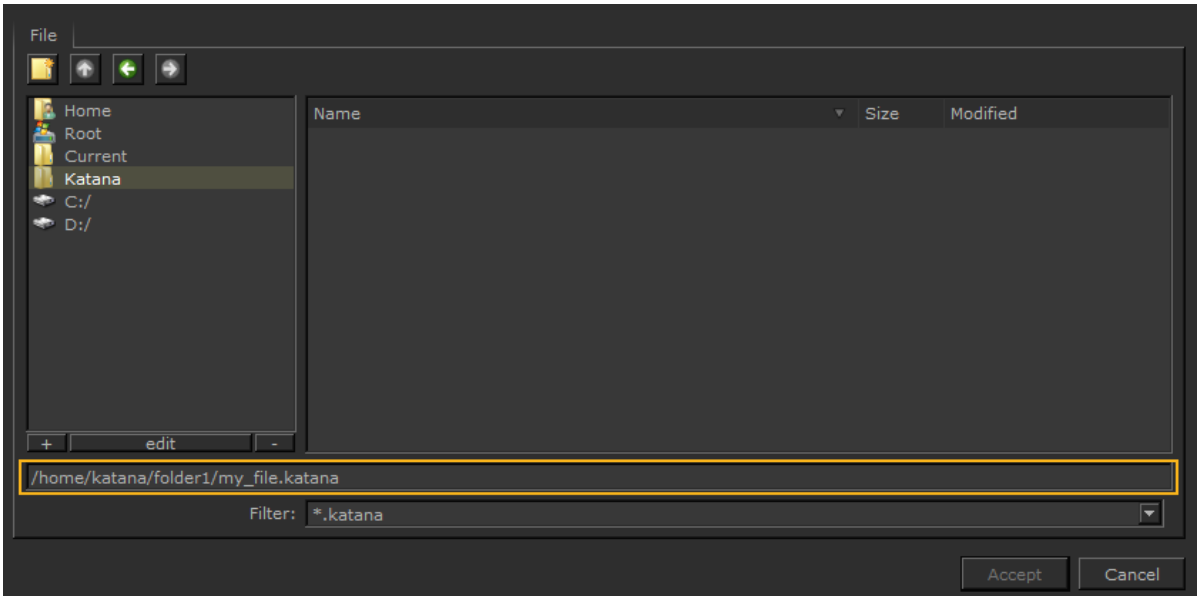
Select **File** > **Save** (or press **Ctrl**+**S**).

If the file has not been saved before, the file browser dialog 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 dialog 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 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 dialog (see Using the File Browser below).
4. Click **Accept**.

Loading a Recently Saved Project

To load a recent Katana project:

1. Select **File** > **Open Recent** > **...** and select from one of the previously saved projects in the dropdown menu.
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 and Exporting a 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 dialog (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.

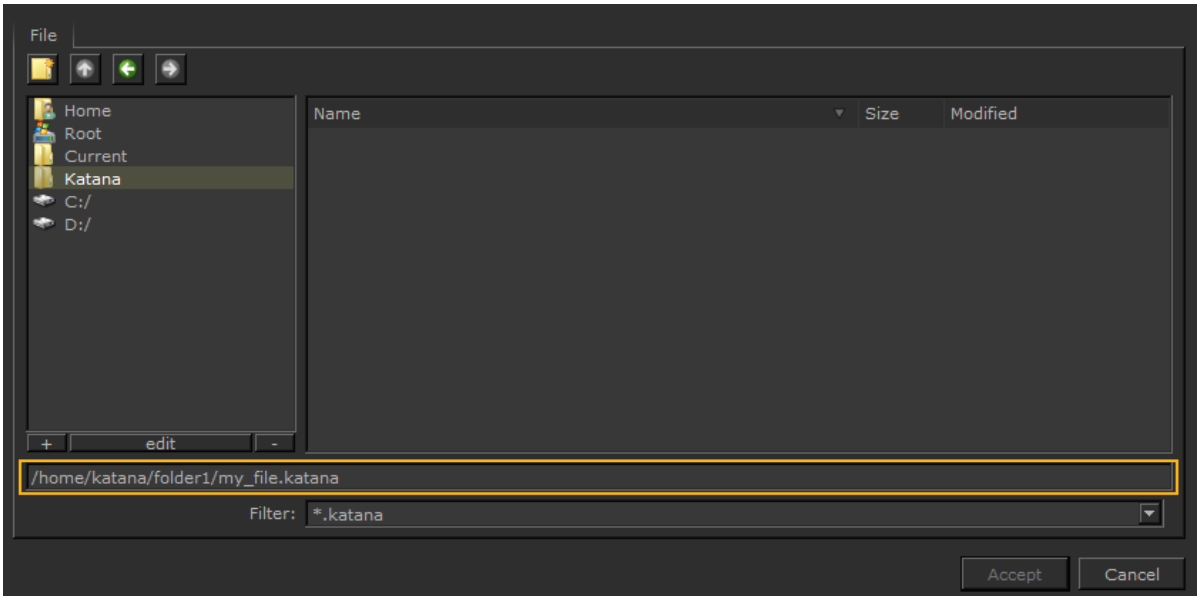
You can also import a Katana project as a LiveGroup. For more information on LiveGroups, and for help on how to import a project as a LiveGroup, refer to the LiveGroups and LiveShadingGroups section.

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 dialog 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

A project's 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	The name of the Katana project you have open. If you are working on an unsaved project, this field is blank.
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.
variables This sub-section is blank if you haven't added any Graph State Variables to your project. Otherwise, the variables and their values are listed.	
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.
compDefaults > motionBlur	

Setting	Description
shutter	The open/close time of the shutter, relative to the current frame. Changing the value in the second field is the primary way to control the amount of motion blur applied.
numSamples	Number of motion blur steps to compute and merge. Render times are proportional to this value.
views > main	
abbreviation	The abbreviation used for the main view when working with stereo controls in the **Monitor** tab.
uicolor	Specify the color of UI elements in the **Monitor** tab when the **main** view is active.
views > left	
abbreviation	The abbreviation used for the left view when working with stereo controls in the **Monitor** tab.
uicolor	Specify the color of UI elements in the **Monitor** tab when the **left** view is active.
views > right	
abbreviation	The abbreviation used for the right view when working with stereo controls in the **Monitor** tab.
uicolor	Specify the color of UI elements in the **Monitor** tab when the **right** view is active.
viewerSettings	
normalsDisplayScale	Changes the size of normals when displayed in the **Viewer** tab.
proxyCacheSize	Number of proxy geometry objects to keep in memory.
viewerSettings > persp	
near	Distance to the near clipping plane for the perspective camera.
far	Distance to the far clipping plane for the perspective camera.

Setting	Description
viewerSettings > ortho	
near	Default distance to the near clipping plane for the orthographic cameras.
far	Default distance to the far clipping plane for the orthographic cameras.
orthoWidth	Default width for the orthographic cameras.
viewerSettings > ortho > front	
near	Distance to the near clipping plane for the front orthographic camera.
far	Distance to the far clipping plane for the front orthographic camera.
orthoWidth	Distance to the width for the front orthographic camera.
viewerSettings > ortho > side	
near	Distance to the near clipping plane for the side orthographic camera.
far	Distance to the far clipping plane for the side orthographic camera.
orthoWidth	Distance to the width for the side orthographic camera.
viewerSettings > ortho > top	
near	Distance to the near clipping plane for the top orthographic camera.
far	Distance to the far clipping plane for the top orthographic camera.
orthoWidth	Distance to the width for the top orthographic camera.
monitorSettings	
overlayColor	Color to use when displaying alpha overlays.

Assets and Asset Managers

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

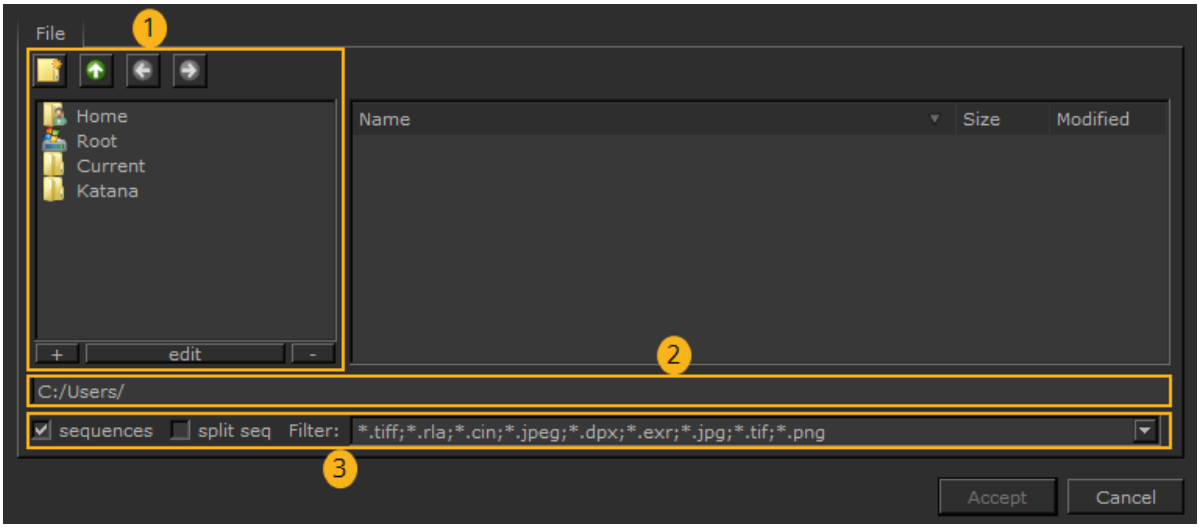
By default, Katana uses the file system to store assets. But Katana has the ability to plug into a studio's asset management system through its Asset Management API. Connecting Katana using this system is beyond the scope of the *Katana User Guide* and you should consult your pipeline manager and the 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. 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

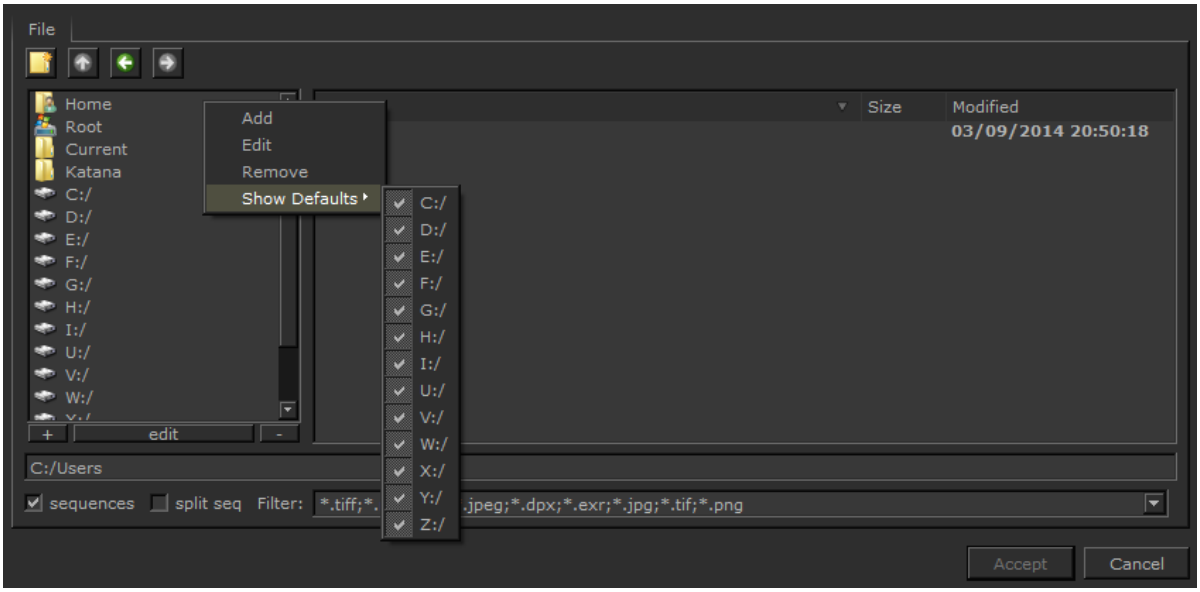
The file browser is the basis for the **File** asset manager.



The following correspond to the numbers on the dialog image above:

1. Navigation controls - let you move through the directory structure, bookmark favorite directories, and create new directory folders.
2. Path name field - displays the current directory path or enter a filepath to the file you want to open.
3. Filter menu - filter what files you can see in the file browser.

Tip: Windows only: You can show/hide the drives that Windows auto-creates by right-clicking the directory list, selecting **Show Defaults**, and checking or unchecking the drive.



Navigation Controls

Use the following controls to navigate between directories:

- Click the **Create New Directory** button to create a new directory at your current position in the file hierarchy.
- Click the **Up One Directory** button to go up one directory closer to the root.
- Click the **Previous Directory** button to go back one directory.
- Click the **Next Directory** button to go forward one directory.
- Click the **+** button to add a directory bookmark.
- Click the **edit** button to edit the name or path name to a bookmark.
- Click the **-** button to remove a directory bookmark.

Path Name Field

The path name field allows you to do the following:

- Navigate to a directory by typing the path name in the field.
- Enter a script name by browsing to a directory path and entering the file name after the displayed path.
- Limit the file list to specific file types by using the **Filter** dropdown menu and **sequences** checkbox.

Filters and Sequences

To use the **Filter** dropdown menu and **sequences** checkbox:

- Select ***.<file extension>*** to display all files of that extension type, for instance ***.png**.
- Select ***** to display all files (except hidden files), regardless of what they're associated with.
- Select **.* *** to display all files, including hidden files.
- Select ***/** to display directory names, but not their contents.
- Check **sequences** to display image sequences as single titles, as in fgelement.####.png 1-50 rather than fgelement.0001.png, fgelement.0002.png, and so on.
- Split incomplete sequences into separate sequences using the **split seq** checkbox.



Note: File sequences with no file extension, for example, fgelement.0001, fgelement.0002, and so on, are not displayed as single titles the first time you view the directory in the file browser. However, they are displayed as single titles once you have navigated to another directory and back again.

Select Multiple Files

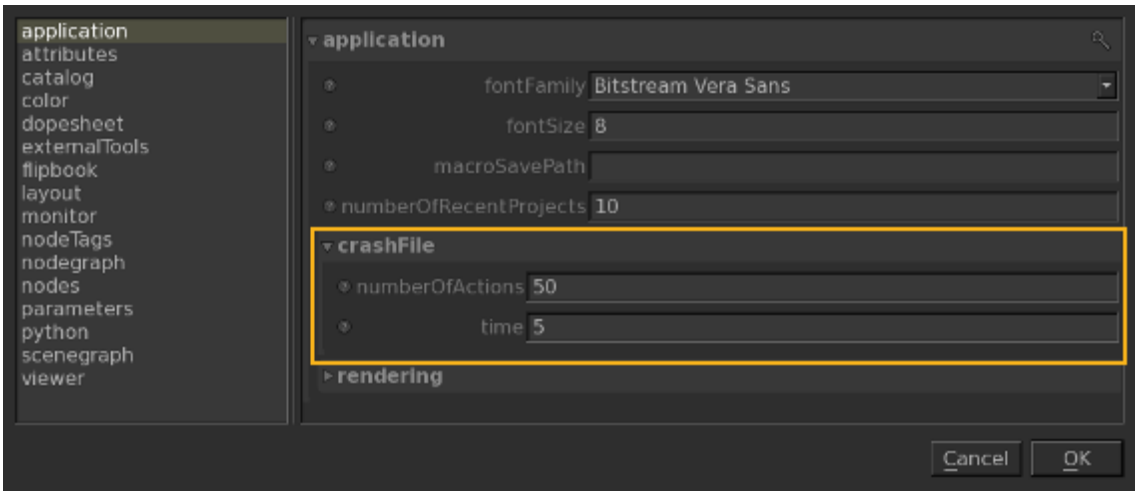
To select multiple files with the file browser:

1. Browse to the folder where the files are located.
2. **Ctrl**+click on all the files you want to open to select them.
3. Click **Open**.

   All the selected files open.

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.



Note: 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

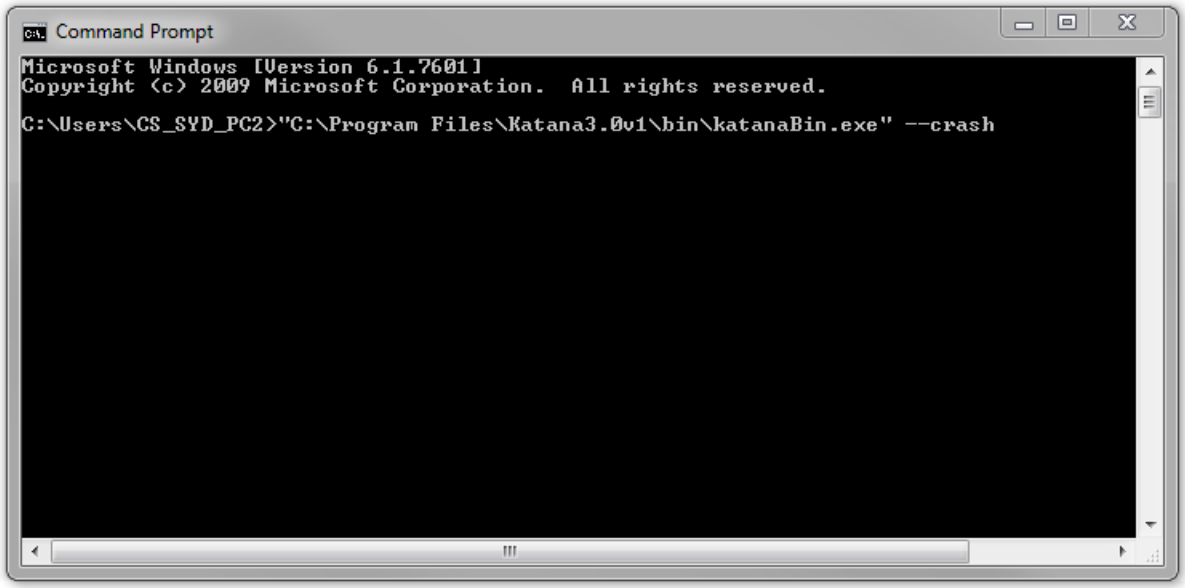
There are two ways to load an autosave file, you can either open it using a command line or you can locate the file within your temporary directory.

Loading an Autosave File using a Command Line

To load an autosave file using a command line:

1. Open a new Command Prompt (cmd) for Windows , or a Terminal for Linux.
2. Specify the path to the Katana executable and add the **--crash** command line option as demonstrated in the commands below.

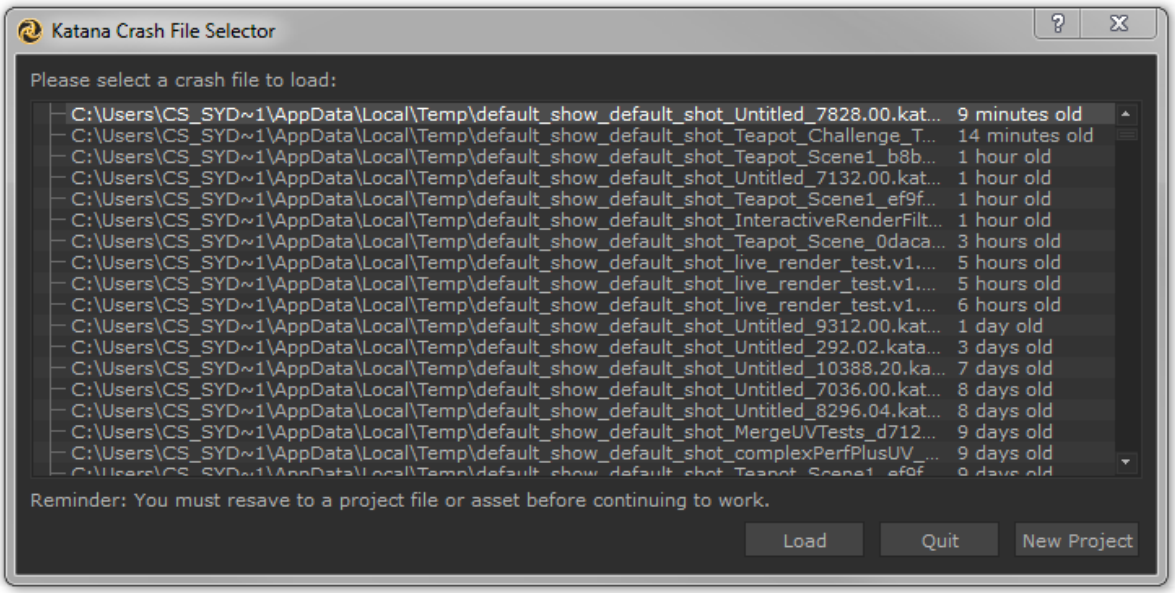
   Windows launch command:
   `C:\Program Files\[KATANA_VERSION]\bin\katanaBin.exe --crash`



   Linux launch command:
   `/opt/Foundry/[KATANA_VERSION]/katana --crash`

3. Execute the command to open Katana.

   Katana will open the **Katana Crash File Selector** window before launching the Katana GUI. This window displays any and all Katana project files found in your temporary directory.



4. The latest autosaved file is the **crashFile** saved when an unexpected exit occurred. Select the latest file and press the **Load** button to load the project in Katana.

   Note: If the crashed scene file had not yet been saved, Katana calls the resulting autosave **Untitled**.

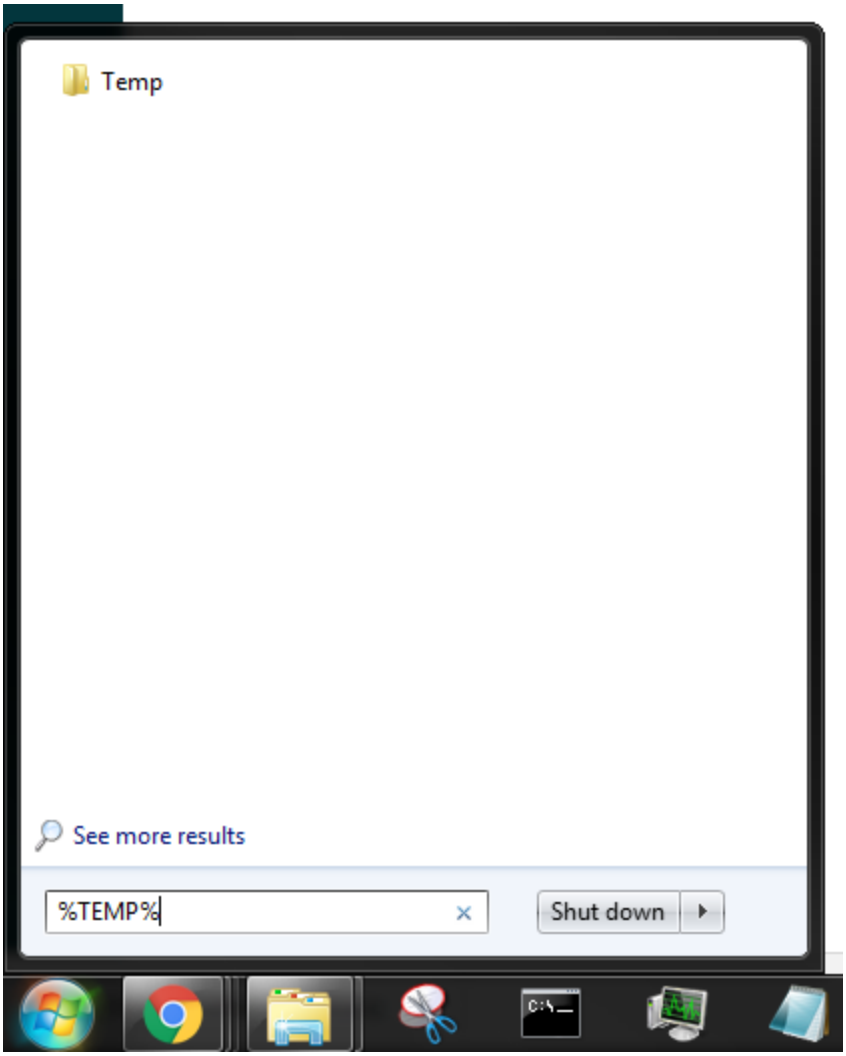
5. Once open, make sure to save the file in the location of the original project before continuing to work. You may want to consider saving this under a different name to the original project, to create a version history of the file.

Loading an Autosave File by Locating it within your Temporary Directory

When Katana creates a **crashFile**, it is saved to the machine's temporary directory. The file can be restored manually from there using the following steps:

Windows:

1. Open the start menu and enter **%TEMP%** in the search bar:



2. Open the **Temp** folder and locate the most recent Katana **crashFile**. The file name should follow a convention similar to the following.
   `default_show_default_shot_<projectFileName>_<crashFileId>.<version>.katana`

> **Note:** If the crashed scene file had not yet been saved, Katana calls the resulting autosave **Untitled**.

3. Open the project in Katana and re-save the file elsewhere.

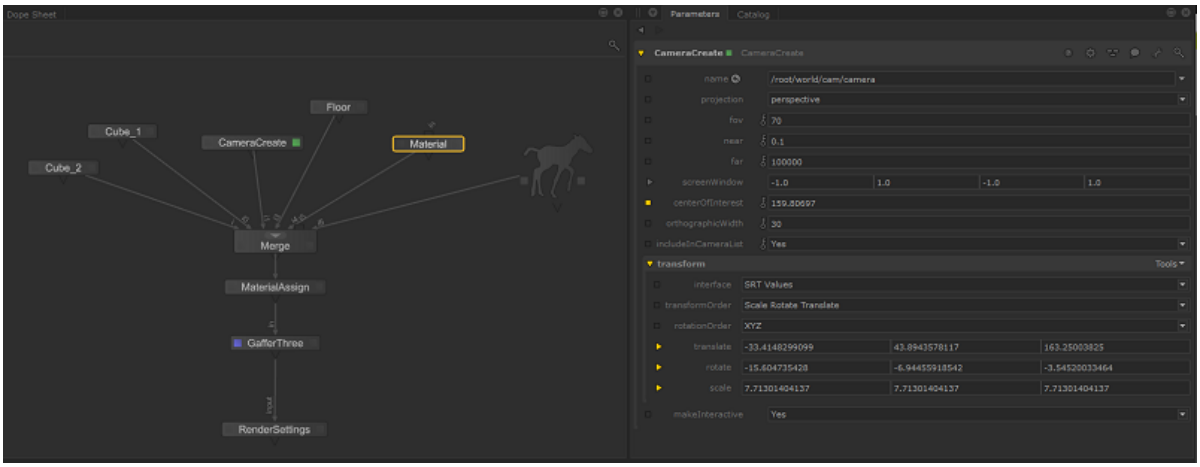
Linux:

1. Navigate to the temporary directory, which should be located at **/tmp**, using either the Terminal or a file browser.

2. Locate the most recent Katana crashFile.
3. Open this in Katana and re-save the project file elsewhere.

Editing the Node Graph

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.



These pages describe how to build and connect nodes within the node graph, and how to edit a node's parameters using the **Parameters** tab.

Navigating Inside the Node Graph

As you set up your recipe, you may need to move between clusters of nodes quickly. Katana offers various navigation methods and shortcuts to help you navigate the **Node Graph** tab quickly.

- Panning - middle-click and drag the mouse pointer over the workspace. The recipe moves with your pointer.
- 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 press **Alt**+left/right-click and drag right. Alternatively, 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 press **Alt**+left/right-click and drag left. Alternatively, 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.

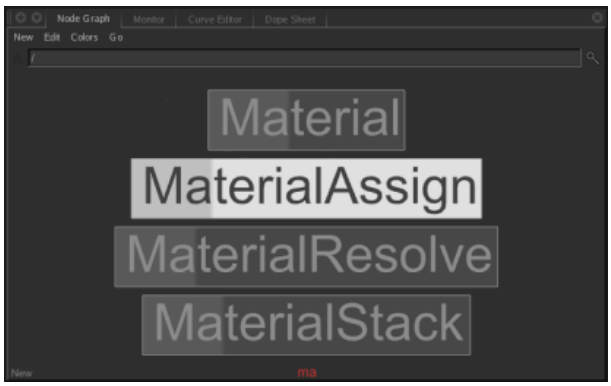
- Fitting selected nodes in the node graph - in the **Node Graph** tab, press **F**. If no nodes are selected, then the entire node tree fills the **Node Graph**.
- Fitting the entire node tree in the node graph - in the **Node Graph**, press **A**.

Adding Nodes

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

To add a node using the **Tab** menu:

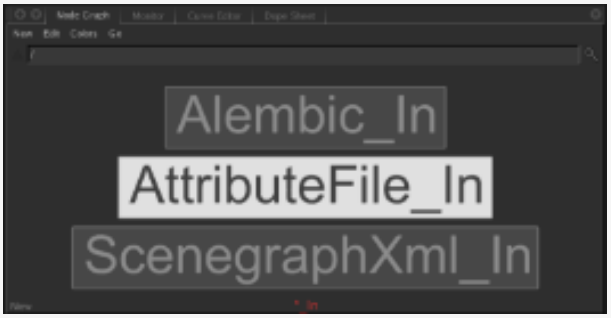
1. With the mouse over the **Node Graph** tab, press the **Tab** button.
   Katana displays a list of all available nodes.
2. Narrow the list of nodes by either:
   - typing the starting letters of the node name, or
   - typing the capital letters that make up the node name (for instance, typing **MA** 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** would give you the following options:



To add 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.

To add 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 node's 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**.

Node Basics

Changing a Node's Name

You can edit node names in a number of different ways:

- In the **Parameters** tab, or in the navigation bar of the **Node Graph** tab while looking inside a Group node, you can rename a node by pressing **F2** or **Return** with the mouse pointer over the node name.
- In the **Node Graph** tab, you can rename a selected node by pressing **F2** or **N**.
- In the **Node Graph** tab, you can rename a node under the mouse pointer by pressing **Return**.

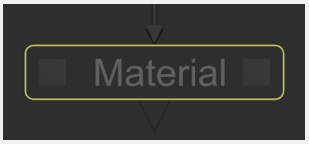
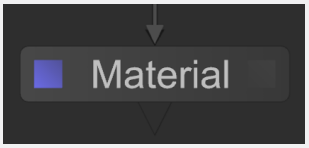
Some nodes derive their name from one of their parameters, for instance **passName** in Render nodes, or **name** in Material nodes. In these cases, you can edit the parameter directly or indirectly by one of the above methods. Changing the node name with either methods updates the name in all instances.

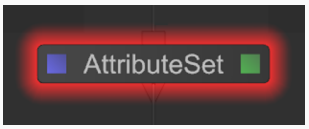
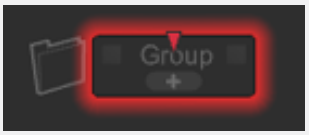
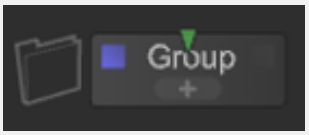
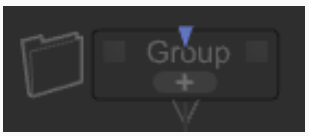


Note: Node names cannot contain spaces. Any spaces or invalid characters that are used in the node name are converted into underscores. For example, the node name "Material Plastic" becomes "Material_Plastic".

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.
	This node is being viewed. The scene graph generated up to this node is currently displayed in the **Scene Graph** tab.

Indicator	What it means
	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.

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.
F2	When a single node is selected, opens a pop-up to edit the name of that node.
N	Opens the right-click node creation menu at the current pointer position. When a single node is selected and visible in the **Node Graph** tab, opens a pop-up to edit the name of that node.
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**	When the pointer is over a node, opens a pop-up to edit the name of that node.
Ctrl+left-click	Reveals and selects the node in the **Node Graph** tab.
Middle-	Drags a representation of the node.

Keyboard Shortcuts	Action
click+drag	
Right-click	Opens the same context menu that can be opened for a node in the **Node Graph** tab.

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

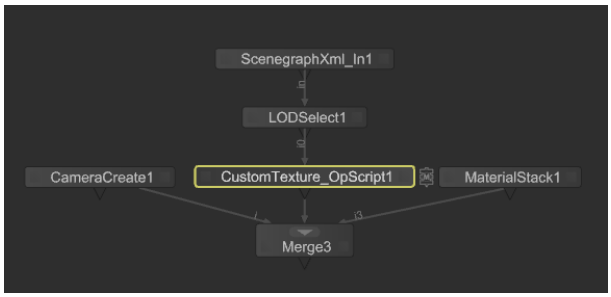
1. Select the node(s).
2. In the **Node Graph**, select **Edit** > **Toggle Disabled State of Selected Nodes** (or press **Alt**+**D**).



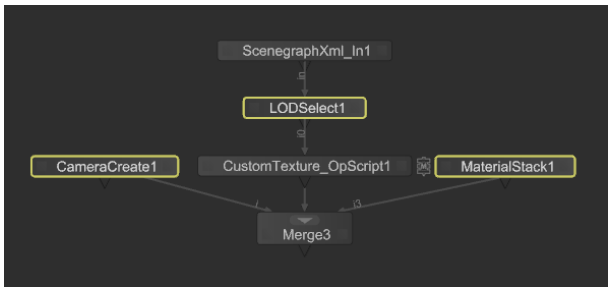
Selecting Nodes

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

To select a single node, click once on the node. To select 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.



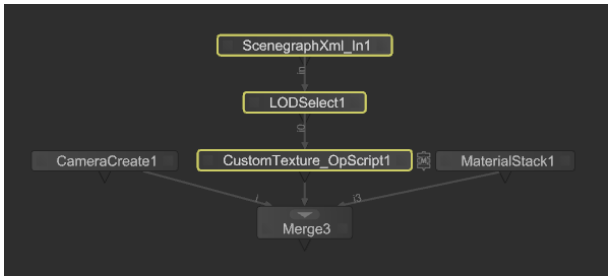
Single node selected.



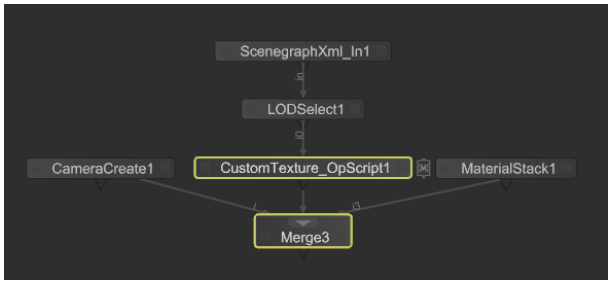
Multiple nodes selected.

To select all nodes upstream of the currently selected node(s), click on a node and press **Ctrl**+**Up Arrow**. Katana selects all nodes that feed data to the selected node.

To select all nodes downstream, of the currently selected node(s), click on a node and press **Ctrl**+**Down Arrow**. Katana selects all nodes downstream from the selected node.



Upstream nodes selected.



Downstream nodes selected.

To add to a selection, **Shift**+click to select more nodes without clearing the current selection.

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. Release the mouse button when the input highlights in yellow.

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.

OR

1. Hover the cursor over the first node you want to connect.
2. Press a number from 1 to 9 to choose the output port at that position.
3. Hover the cursor over the second node and press a number from 1 to 9 to connect to the input port at that position.

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.

Merging Nodes

You can merge any number of selected nodes in the **Node Graph** tab either through the tab's **Edit** menu or the available keyboard shortcut. Merging selected nodes creates a Merge node that links up the selected nodes' outputs to the inputs of the Merge node. A Merge node with a single input is effectively a no-Op node.

To merge selected nodes, either press **M** when in the **Node Graph** tab or select **Edit** > **Merge Selected Nodes** from the **Node Graph** tab's menu bar.

Removing, Replacing, and Deleting Nodes

Removing a Node

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

- remove the inputs/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.

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

To extract a node, removing all the connections to the node without deleting it:

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.

Replacing Nodes

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

Deleting Nodes

To delete selected nodes

1. Select the node(s) you want to delete.
2. Press **Delete**.

   Katana removes the node(s) from the scene.

To delete 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.



Warning: Katana will attempt to bridge the connections between adjacent connected nodes when a node is deleted, which can sometimes lead to unwanted streams of data that may be incompatible. Holding the **Ctrl** key while deleting a node will disable bridging connections and stop any upstream or downstream nodes from being automatically reconnected.

Copying, Pasting, and Cloning Nodes

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.

To copy nodes to the clipboard:

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

To paste nodes from the clipboard:

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

Katana adds the nodes to the scene.

To cut nodes from the **Node Graph**

1. Select the node(s) 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.

Grouping 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 **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, in that 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:

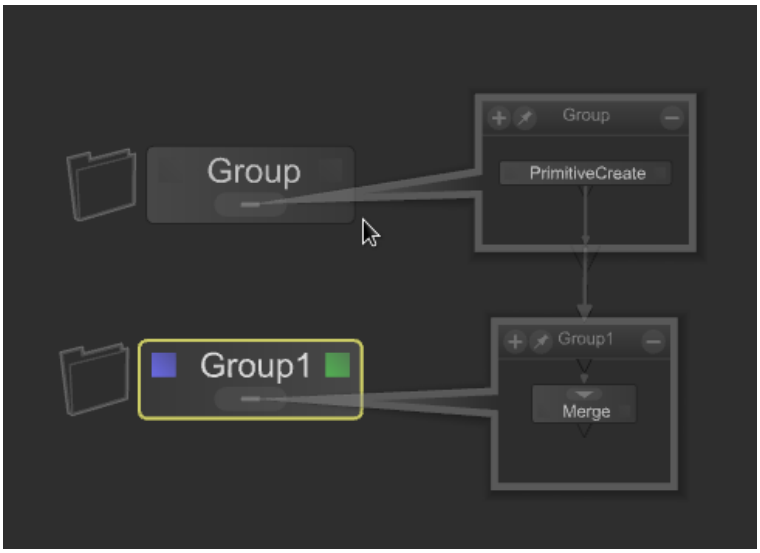
- Press **Ctrl**+**Shift**+**Backspace**,
- From the **Node Graph** tab's menu, navigate to **Go** > **To Root**, or
- Click the **Go To Root** button in the breadcrumbs bar.

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 **Help** > **Developer 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.

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** 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**.

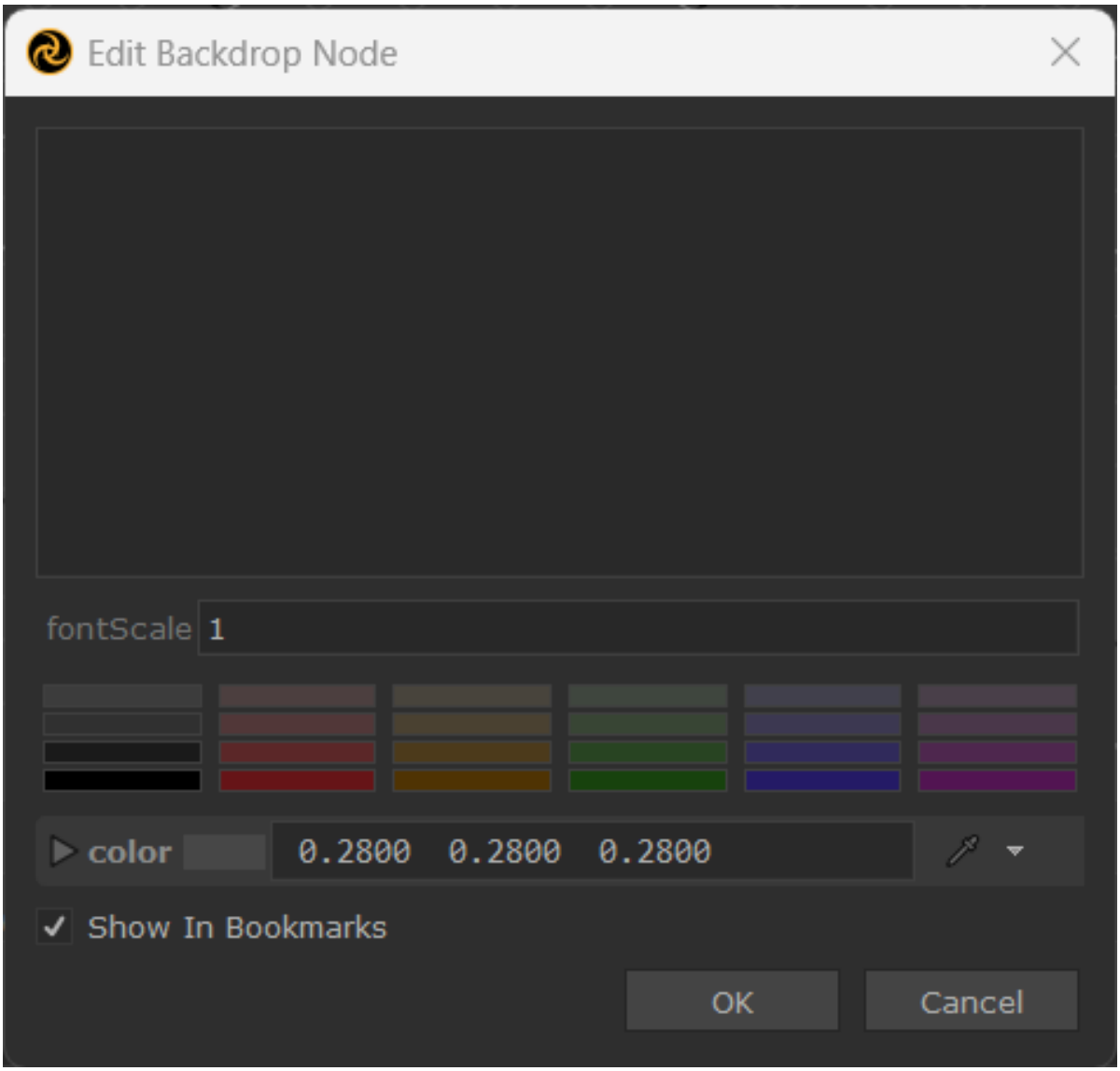
   If you select a Backdrop node with the selected nodes, Katana uses that Backdrop node, otherwise a new Backdrop node is created.

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 **EditBackdrop 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).

The Z ordering of Backdrop nodes can now be controlled by the **[** and **]** keyboard shortcuts, to bring Backdrop nodes to the back and front respectively. As part of this feature, the Send to Back checkbox has been removed from the Edit Backdrop Node dialog.

3. Click **Ok** to save changes.

You can also 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**.

If you want to select all nodes within the bounds of a Backdrop node (as well as the node itself), you can **Ctrl**+click 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**.

Dot Nodes

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 **.** (period) to create a Dot node.

> **Tip:** You can also create the Dot node in the same way as any other node using the **Tab** menu, **New** menu, or right-click menu.

3. Drag the Dot node as necessary to reposition the connections.

Advanced Display Options

You can change how a node is displayed to improve clarity and readability and to provide additional information about a node's behavior. You can also reduce the contrast around nodes and their connections by selecting **Edit** > **Draw Graph with Low Contrast** , in the **Node Graph**. You can do this in conjunction with dimming unconnected nodes, described in more detail below.

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.

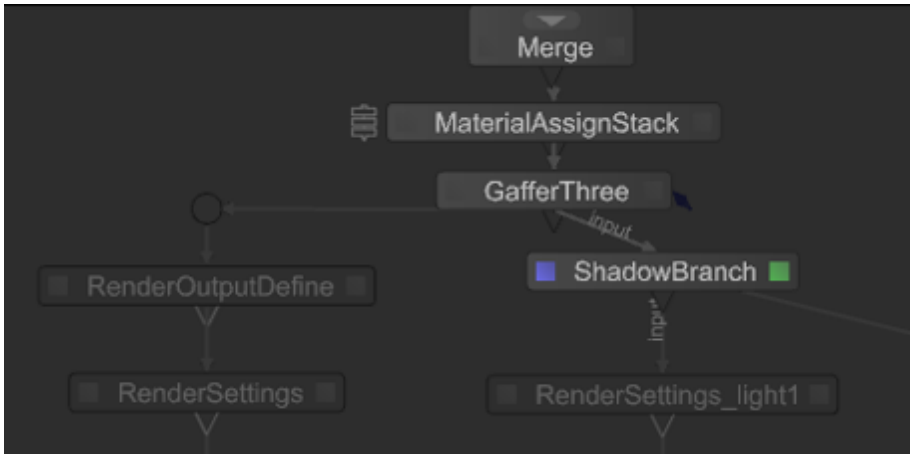
To change a node's background color to a custom color:

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+.** (period) in the **Node Graph** tab.

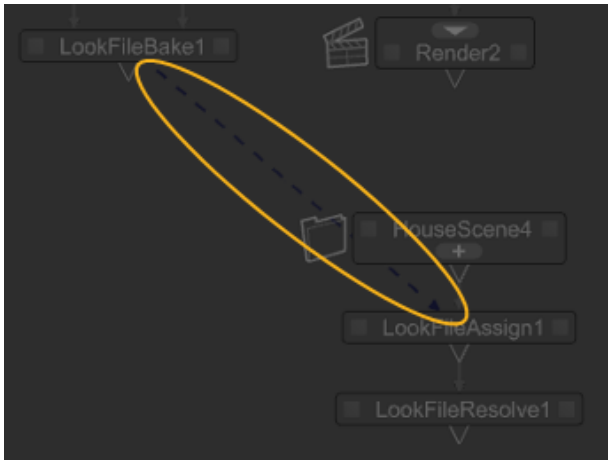


For instance, with the Switch node, you can use the **Dim Nodes Not Contributing to Viewed Node** option in the **Edit** menu of the **Node Graph** tab to visualize which portion of the node graph has been selected by the Switch node.

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.

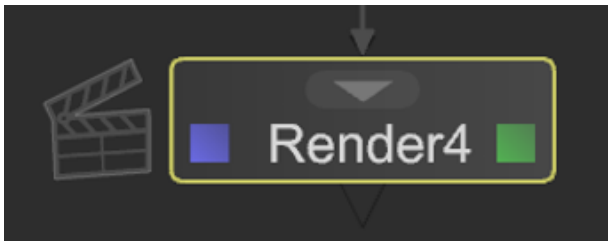
Displaying Nodes Linked by an Expression

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.



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!

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. Please refer to the *Developer Guide* for more information.
- A curve: only available for numeric inputs. See Animation.

Node Parameter Basics

Default Parameters Tab Icons

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



1. Open/Close the node grouping.
2. The name of the node, which can be changed in the parameters or by clicking on the name.

3. The type of node selected.
4. Toggle the node tooltip.
5. Toggle the node shelf.
6. Toggle the graph state variables window.
7. Toggle user comments for the node.
8. Toggle the parameters menu.
9. Toggle the parameter search window.

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 **Tabs** > **Parameters**.

Accessing a Node's Parameters

To edit a node's parameters, they need to be in the **Parameters** tab. To do this, select the node(s) whose parameters you want to edit, then:

- In the **Node Graph**, select **Edit** > **Edit Selected Nodes** (or press **Alt**+**E**).
- Hover the mouse pointer over the node you wish to edit and press **E** .
- Click within the faint square to the right of a node.
- Double-click on a node. This also sets the current scene graph view to that node. See Using the Scene Graph for more information.

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



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. Please refer to the *Developer Guide* for more information.
- A curve: only available for numeric inputs. See Animation for more information.

Each parameter type has a control associated with it, and listed below are a few ways you can change the common parameter types.

Changing a Numeric Value

You can change a numeric value by:

- Double-clicking in the field to select the whole value and entering a new value in the input field.



Note: Positive and negative numbers are supported. You can also enter integers (12345), Floating/Double points (12.345), scientific notations (1e2345), and hexadecimal numbers (0x9abcd).

Tip: Katana also allows you to enter formulas into fields, making it easy to do quick calculations. For example, if you wanted to halve a value of 378, you could simply type **378/2** into a field and press **Enter** to get 189.

You can increment or decrement field values by hundreds, tens, tenths, hundredths, and so on. The magnitude of change depends on the initial position of your cursor. For example if you wanted to increment the initial value of **20.51** by ones, you would insert your cursor before the **0**.

To increment or decrement a field value:

- Click to insert the cursor just prior to the digit you want to increment or decrement and press the **up arrow** to increment by one unit, or the **down arrow** to decrement by one unit.



Tip: You can also increment and decrement values using the mouse wheel (if available).

- Click and drag on the parameter name, also known as scrubbing. Dragging to the left decreases the value, and dragging to the right increases.



Note: If the **stickyDrag** option has been enabled in the **nodegraph** preferences, you can click on the parameter label and move the mouse left to decrease the value or right to increase it.
To access the **Preferences** dialog, either select **Edit** > **Preferences** from the main menu bar or select **Edit** > **Preferences** from the **Node Graph** tab's menu bar.

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,

- 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.

Common Parameter Widgets

These widget groups are common to many nodes in Katana and are outlined here. For more information regarding the addition of user parameters and specific widget types, refer to Widget Types and Adding User Parameters.

Asset and File Path Widget Types

The **Asset** (assetIdInput) and **File Path** (fileInput) widget types allow you to navigate to assets and files on your file system. Several node types that ship with Katana use the **Asset** and **File Path** widget types for parameters of various names, for example: **abcAsset**, **file**, **filePath**, **lookfile**, **procedural**, **saveTo**, and **source**.

Parameters that use the **Asset** and **File Path** widget types can be found on the following types of nodes:

- Alembic_In
- AttributeFile_In
- CameraImagePlaneCreate
- GafferThree
- ImageCoordinate
- ImageRead
- LiveGroup
- LookFileAssign
- LookFileBake
- LookFileGlobalsAssign
- LookFileMaterialsIn
- LookFileMaterialsOut
- LookFileMultiBake
- LookFileOverrideEnable
- Material
- RendererProceduralArgs
- RenderOutputDefine

Menu Command	Description
▾	
Browse...	Brings up the file browser or your studio's asset management browser and enables you to select the asset to use.
Set Node Name From Path	Changes the name of the node to match the filename but without the path or extension.

Attribute Name Widget Type

The **Attribute Name** (attributeName) widget type allows you to type, or drag and drop attributes from the **Attributes** tab, onto parameters of type **String** in the user parameters. When dropping a dragged attribute,

the target parameter's value is set to the name of the dropped attribute instead of the value of the dropped attribute. Names of ancestor group attributes are separated by dots, for example, **xform.translate**.

Parameters that use the **AttributeName** widget type can be found on the following types of nodes:

- AttributeSet

Menu Command	Description
Text field	
newParameter	Type, or drag and drop attributes from the **Attributes** tab to set the parameter's value to the name of the attribute.

Attribute Type Widget Type

The **Attribute Type** (attributeType) widget type allows you to select from a drop-down menu, or drag and drop attributes from the **Attributes** tab, onto parameters of type **String** in the user parameters. When dropping a dragged attribute, the target parameter's value is set to the name of the dropped attribute's type, for example, **float**.

Parameters that use the **AttributeType** widget type can be found on the following types of nodes:

- AttributeSet

Menu Command	Description
Text field	
newParameter	Select an attribute type from the drop-down menu, or drag and drop attributes from the **Attributes** tab, to set the parameter's value to the attribute type.

CEL Statement Widget Type

The **CEL Statement** (cel) widget type allows you to build and edit CEL statements that are stored in string parameters on nodes. Several node types that ship with Katana use the **CEL Statement** widget type for

parameters of various names, for example: **CEL**, **cel**, **toCel**, **fromCel**, **celSelection**, **disableAt**, **exclusivity**, **lights**, **objects**, **off**, and **on**.

Parameters that use the **CEL Statement** widget type can be found on the following types of nodes:

- AttributeCopy
- AttributeEditor
- AttributeFile_In
- AttributeSet
- CollectionCreate
- GafferThree
- GenericOp
- LightLink
- LightLinkEdit
- LightLinkSetup
- LocationGenerate
- LodSelect
- LodValuesAssign
- LookFileAssign
- Material
- OpScript
- Prune
- RendererProceduralArgs
- RendererProceduralAssign
- ReverseNormals
- ScenegraphObjectSettings
- VelocityApply
- ViewerObjectSettings
- VisibilityAssign

Menu Command	Description
Add Statements	• **Paths** - Adds a **Paths** list to this CEL parameter. • **Collections** - Adds a **Collections** list to this CEL parameter. • **Custom** - Adds a **Custom** parameter to this CEL parameter. • **Append Scene Graph Selection** - Adds a **Paths** list to this CEL parameter and places selected scene graph locations in the new list. • **Replace With Scene Graph Selection** - Removes any parameters within this CEL parameter and creates a new **Paths** list and populates it with any selected scene graph locations. • **Copy CEL Statement As Text** - Copies this CEL statement to the clipboard. • **Paste CEL Statement** - Removes any parameters within this CEL parameter and pastes the CEL statement in the clipboard to this parameter. • **Replace With Parameter Expression** - Converts the current CEL parameter into an expression.
Paths > Action	
Add Scenegraph Selection	Adds the currently selected scene graph location to this list.
Remove Scenegraph Selection	Removes the currently selected scene graph location from this list.
Remove Selected Paths	Removes the path(s), selected in this **Paths** list, from this list.

Menu Command	Description
Select All	Selects all the paths in this list.
Select Selected Paths In Scenegraph	Selects the scene graph locations of the selected paths in this list.
Copy Selected Paths to Clipboard	Copies the selected paths from this list to the clipboard.
Show Extended View...	Brings up a dialog with the contents of this **Paths** list.
Collections > Action	
Add Collections From Scenegraph Selection...	Brings up a dialog box with a list of the collections from the currently selected scene graph locations. You can then select from these collections to add them to this list.
Add Scene Root Collections...	Brings up a dialog box populated with the collections currently on **/root**. You can then select from these collections to add them to this list.
Remove Selected Paths	Removes the selected collection(s) from this list.
Select All	Selects all the collections in this list.
Copy Selected Paths to Clipboard	Copies all the selected collections and their paths to the clipboard.
"Find And Select" Selected Items...	
Union dropdown	• **Union** • **Difference** • **Intersect** Does not occur in all nodes with CEL widgets. Only occurs on additional statements added to the widget after an initial statement.

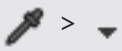
Color Widget Type

The **Color** (color) widget type allows you to pick a color by specifying RGB or RGBA component values directly in the **Parameters** tab, or through a color picker dialog. Several node types that ship with Katana use

the **Color** widget type for parameters of various names, for example: **bottomLeft**, **bottomRight**, **color**, **constantColor**, **fadeToColor**, **gamma**, and **previewColor**.

Parameters that use the **Color** widget type can be found on the following types of nodes:

- GafferThree
- ImageBackgroundColor
- ImageChannels
- ImageCheckerboard
- ImageColor
- ImageClamp
- ImageContrast
- ImageFade
- ImageGamma
- ImageInvert
- ImageLevels
- ImageRamp
- ImageText
- ImageThreshold
- LightCreate
- ViewerObjectSettings

Menu Command	Description
color	The color (RGBA values) for the given parameter.
	Picks the color (RGBA) values of the selection.
	
Average	Sets the color picker to use the average values.
Min	Sets the color picker to use the minimum values.
Max	Sets the color picker to use the maximum values.
Front	Sets the color picker to use the front values.
Back	Sets the color picker to use the back values.
Auto-Disable Upon Release	Toggle the ability to automatically disable the picker on mouse-button release.
color > RGB	
red	Sets the red value of the pixels.
green	Sets the green value of the pixels.
blue	Sets the blue value of the pixels.

alpha	Sets the alpha value of the pixels.
color > HSL	
hue	Sets the hue of the pixels.
saturation	Sets the saturation of the pixels.
lightness	Sets the lightness of the pixels.
alpha	Sets the alpha value of the pixels.
color > HSV	
hue	Sets the hue of the pixels.
saturation	Sets the saturation of the pixels.
value	Sets the value of the pixels.
alpha	Sets the alpha value of the pixels.
color options continued	
Enable Display Transform	Toggles gamma correction in the color picker, which is especially useful when working with OCIO.
Restrict RGBA Components	Restricts the alpha to 0,1 and the color channels to 0,a.

Common 2D Node Widget Type

The **Common 2D Node** (node2d) widget type allows you to pick channels that are affected by a particular 2D node and specify masking parameters. This widget type is not exposed for use as a custom user parameter.

The **Common 2D Node** widget type can be found in each Image node, except for Image BackgroundColor, ImageChannels, ImageCheckerboard, ImageColor, ImageCoordinate, ImageCrop, ImageRamp, ImageRead, ImageText, and ImageWrite.



Note: Not all of the Image nodes have the **Mask** parameter.

Menu Command	Description
▼ [view]	Specify whether the controls are set for the **main**, **left**, or **right** views, or set the controls to **Enable All**.
	When a specific component (R, G, B, or A) is enabled, the controls affect only that component.
mix	Dissolves between the **bg** image at 0 and the full merge effect at 1.
Mask ▼	
channel	The channel from the **out_mask** input to use as a mask: • **R** - use the red channel as the mask. • **G** - use the green channel as the mask. • **B** - use the blue channel as the mask. • **A** - use the alpha channel as the mask. By default, the merge is limited to the non-black areas of the mask.
invert	Inverts the use of the mask channel so that the merge is limited to the non-white areas of the mask.
fringe	When enabled, the mask is modified so that, by default, the merge is limited to the fringe (semi-transparent areas). This is common alpha treatment, which modifies a normal mask such that it only affects the fringe (semi-transparent) areas.

New Scene Graph Location Widget Type

The **newScenegraphLocation** (newScenegraphLocation) widget type allows you to specify the path of a scene graph location that is to be created by the respective node. It is used, for example, for the **name** parameter of CameraCreate nodes.

This widget type is not exposed for use as a custom user parameter, but can be accessed by setting the widget type hint to **newScenegraphLocation**. Several node types that ship with Katana use the **newScenegraphLocation** widget type for parameters of various names, for example: **location**, **locations**, and **name**.

Parameters that use the **newScenegraphLocation** widget type can be found on the following types of nodes:

- Alembic_In
- CameraCreate
- ImageCoordinate
- LightCreate
- PonyStack
- PrimitiveCreate
- TeapotCreate

Menu Command	Description
Parent to Scenegraph Selection	Sets the parent location of the object created to be the current scene graph selection.
Parent to /root/world/geo/...	Sets the parent location of the object created to be **/root/world/geo/**.
Parent to /root/world/lgt/...	Sets the parent location of the object created to be **/root/world/lgt/**.
Parent to /root/world/cam/...	Sets the parent location of the object created to be **/root/world/cam/**.
Select In Scene Graph	Selects the location specified by this parameter in the **Scene Graph** tab. **Note:** This option may be included in both the ▼ dropdown menu and as an icon to the right of the parameter name, or may only be present in one of these locations.

Locations Widget Type

The **Locations** (locations) widget type allows you to select scene graph locations by paths or expressions. It is not an exposed widget type for use as a custom user parameter. Out of all node types that ship with Katana, only Isolate nodes use the **Locations** widget type, namely for their **isolateLocation** parameter.

Menu Command	Description
Path	Adds another path to this parameter's list of paths.

Expressions	
Append Scene Graph Selection	For each selected scene graph location, a new path is added to this parameter's list of paths and populated with the location.
Replace with Scene Graph Selection	Removes all existing paths and replaces them with paths populated with the currently selected **Scene Graph** tab locations.
Append Node Graph Node Locations	For each selected **Node Graph** node, a new path is added to this parameter's list of paths and an expression that links the scene graph location created by that node to the path.
Replace with Node Graph Node Locations	Removes all existing paths and replaces them with a path for each selected **Node Graph** node and links the scene graph location created by that node to the path.
Append Node Graph Locations as Parameter Expressions	
Replace Node Graph Locations as Parameter Expressions	
Clear All	Removes all paths from this parameter.

Look File Pass Name Widget Type

The **Look File Pass Name** (lookfilePassname) widget type allows you to set the pass name to use from a chosen Look File. This widget type is not exposed for use as a custom user parameter, but can be accessed by setting the widget type hint to **lookfilePassname**. Node types that ship with Katana use the **Look File Pass Name** widget type for the **passName** parameter.

Parameters that use the **Look File Pass Name** widget type can be found on the following types of nodes:

- LookFileMaterialsIn
- LookFileOverrideEnable
- LookFileResolve

Menu Command	Description
Choose Look File Pass from Selection...	

Rectangle Widget Type

The **Rectangle** (rectangle) widget type allows you to specify rectangular bounds to use by a node. This widget type is not exposed for use as a custom user parameter. Several node types that ship with Katana use the **Rectangle** widget type for parameters of various names, for example: **bounds**, **resolution**, or **rect**.

Parameters that use the **Rectangle** widget type can be found on the following types of nodes:

Menu Command	Description
bounds or **rect** >	
Copy from Monitor ROI	
Copy to Monitor ROI	

Scene Graph Location Widget Type

The **Scene Graph Location** (scenegraphLocation) widget type allows you to specify the path of an existing scene graph location that a node is meant to work with. Several node types that ship with Katana use the **Scene Graph Location** widget type for parameters of various names, for example: **baseLocation**, **cameraLocation**, **location**, **locations**, **path**, **paths**, **sourceLocation**, and **targetPath**.

Parameters that use the **Scene Graph Location** widget type can be found on the following types of nodes:

- AimConstraint
- AttributeCopy
- AttributeSet
- BillboardConstraint
- BoundsAdjust
- GafferThree
- HierarchyCopy
- InfoCreate
- Isolate
- LightLink
- OpScript
- OrientConstraint
- ParentChildConstraint
- PointConstraint
- ReflectionConstraint

- CameraClippingPlaneEdit
- CameraImagePlaneCreate
- CameraScreenWindowConstraint
- ClippingConstraint
- CollectionCreate
- ConstraintCache
- CoordinateSystemDefine
- DollyConstraint
- FaceSetCreate
- FOVConstraint
- LightLinkEdit
- LightLinkSetup
- LightLinkEdit
- LocationCreate
- LookFileBake
- LookFileMultiBake
- Material
- NetworkMaterialInterfaceControls
- NetworkMaterialParameterEdit
- NetworkMaterialSplice
- Rename
- RendererProceduralArgs
- RendererProceduralAssign
- RenderOutputDefine
- RenderSettings
- ScreenCoordinateConstraint
- Transform3D
- TransformEdit

Menu Command	Description
Adopt Scenegraph Selection	The currently selected **Scene Graph** tab location is used to populate the parameter.
Adopt Selected Nodegraph Node	Creates an expression from the currently selected **Node Graph** node linking the scene graph location created by that node to this parameter.
Select In Scenegraph	Selects the location specified by this parameter in the **Scene Graph** tab. **Note:** This option may be included in both the dropdown menu and as an icon to the right of the parameter name, or may only be present in one of these locations.
Adjust Path Relative To 'basePath'	Converts the current **targetPath** to a path relative to the **basePath**. If the **targetPath** is an expression, it is converted to a constant. **Note:** This option does not appear in all instances and may only be available when both the **basePath** and **targetPath** parameters exist for a node.

Scene Graph Locations Widget Type

The **Scene Graph Locations** (scenegraphLocationArray) widget type allows you to specify a list of paths of existing scene graph locations that a node is meant to work with. Several node types that ship with Katana use the **Scene Graph Locations** widget type for parameters of various names, for example: **destinationLocations**, **lightPaths**, **locations**, **paths**, and **rootLocations**.

Parameters that use the **Scene Graph Locations** widget type can be found on the following types of nodes:

- AimConstraint
- AttributeSet
- BillboardConstraint
- ClippingConstraint
- ConstraintCache
- ConstraintListEdit
- DollyConstraint
- FOVConstraint
- HierarchyCopy
- InfoCreate
- LightLink
- LightListEdit
- LocationCreate
- LookFileBake
- LookFileMultiBake
- PointConstraint
- ScreenCoordinateConstraint

Menu Command	Description
Path	Adds another path to this parameter's list of paths.
Append Scenegraph Selection	For each selected scene graph location, a new path is added to this parameter's list of paths and populated with the location.
Replace with Scenegraph Selection	Removes all existing paths and replaces them with paths populated with the currently selected **Scene Graph** tab locations.
Append Nodegraph Node Locations	For each selected **Node Graph** node, a new path is added to this parameter's list of paths and an expression that links the scene graph location created by that node to the path.
Replace with Nodegraph Node Locations	Removes all existing paths and replaces them with a path for each selected **Node Graph** node and links the scene graph location created by that node to the path.
Clear All	Removes all paths from this parameter.

Find Instances beneath Scene Graph Selection...	**Note:** This option is only found on the **rootLocations** parameter.

Transform Controls Widget Type

The **Transform Controls** widget type allows you to manipulate the transformation matrix. This widget type is not exposed for use as a custom user parameter. Node types that ship with Katana use the **Transform Controls** widget type for the transform parameter.

Parameters that use the **Transform Controls** widget type can be found on the following types of nodes:

- CameraCreate
- GafferThree
- LightCreate
- PonyStack
- PrimitiveCreate
- TeapotCreate

Menu Command	Description
transform	
interface	Sets the transform control layout: • **SRT Values** - exposes the scale, rotation, and translation controls. • **Transform Matrix** - exposes a matrix to control transformations. **Note:** If you select **Transform Matrix**, the **translate**, **rotation**, and **scale** fields are replaced by a **matrix** field instead.
transformOrder	Sets the order in which transforms are applied: **Scale Rotate Translate**, **Scale Translate Rotate**, **Rotate Scale Translate**, **Rotate Translate Scale**, **Translate Scale Rotate**, **Translate Rotate Scale**.
rotationOrder	Sets the order in which rotation is applied: **XYZ**, **XZY**, **YXZ**, **YZX**, **ZXY**, **ZYX**.
transform > interface: SRT Values	

Menu Command	Description
translate	Controls camera translation on the xyz axes.
rotate	Controls camera rotation on the xyz axes.
scale	Controls camera scale on the xyz axes.
transform > interface: Transform Matrix	
matrix	Controls transformations using a matrix in place of individual SRT controls. **Note:** This field is only available if you have selected **Transform Matrix** in the **interface** field.

Transform Tools Widget Type

The **Transform Tools** widget type allows you to manipulate transformation data for scene graph locations. This widget type is not exposed for use as a custom user parameter. Node types that ship with Katana use the **Transform Tools** widget type for the transform parameter.

Parameters that use the **Transform Tools** widget type can be found on the following types of nodes:

- CameraCreate
- GafferThree
- LightCreate
- PrimitiveCreate
- TeapotCreate

Menu Command	Description
transform > Tools	
Snap to Position of Scene Graph Selection	Moves the position of the light to the position of the item selected in the **Scene Graph** tab.
Copy Scene Graph Selection World Transform	Copies the world SRT values of the item selected in the **Scene Graph** tab into the translate, rotate, and scale parameters under **Object** tab > **transform**.

Menu Command	Description
Copy Scene Graph Selection Local Transform	Copies the local SRT values of the item selected in the **Scene Graph** tab into the translate, rotate, and scale parameters under **Object** tab > **transform**.
Fit to Bounds of Scene Graph Selection	Fits the light to the bounds set by the item selected in the **Scene Graph** tab.
Register to Scene Graph Camera	Places an object at a specified distance from a camera that is selected in the **Scene Graph** tab, oriented to face the camera, and scaled to fit the camera's screen window exactly. This is designed for use with primitive planes and may set unexpected transforms on other object types. This option is only available if a camera or light is selected in the **Scene Graph** tab.
Reset Transform	Resets any previous transforms, bringing the light back to the origin (0,0,0).

Parameter State Badges

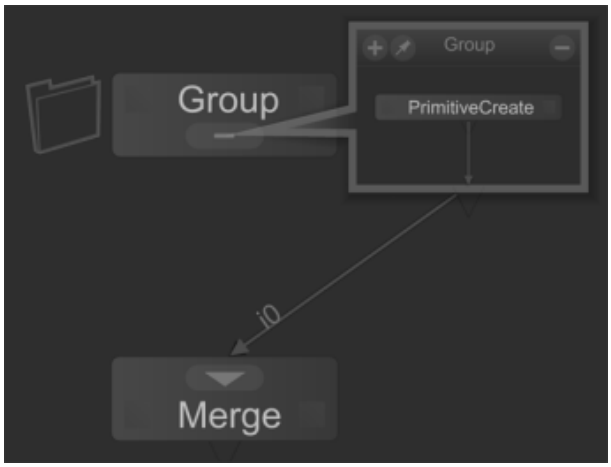
Some parameters, and all attributes, have an icon to help you determine how the current value is being assigned. These are referred to as state badges.

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.

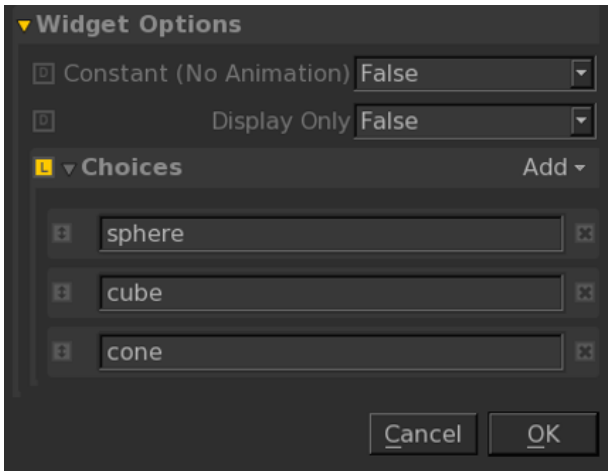
Adding User Parameters

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. User parameters can also drive parameters on nodes in the recipe outside of the group or macro. It can be useful to present the user of a group or macro with a series of known, valid choices in the form of a pop-up menu. You can create this, as shown below, by editing the user parameters.

1. Start with a recipe consisting of a Group node with a child 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, click > **Edit User Parameters**.
4. On the **user** parameter, click **Add** > **String**.
   A new user parameter of type **string** is created.
5. Change the widget type of your user parameter to **Popup Menu** by clicking the menu on the user parameter and selecting **Widget Type** > **Popup Menu**.
   This changes the user parameters widget type to pop-up menu, where each entry in the menu is a string.
6. Edit the pop-up menu to add new entries, each corresponding to a valid PrimitiveCreate node type. To do this, in the menu of the new parameter, click **Widget Options...**. In the resulting widget options dialog, click **Add** > **New Entry**, and name them "sphere", "cube", and "cone".



7. Link the user parameter pop-up menu to the **type** parameter of the PrimitiveCreate node by right-clicking on the user parameter pop-up menu and clicking **Copy**, then selecting 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.

> **Tip:** If you want the user parameters to be shown at the top level on the Group node, you can click > **Show User Parameters At Top Level** in the Group node's parameters before you finish editing the user parameters.

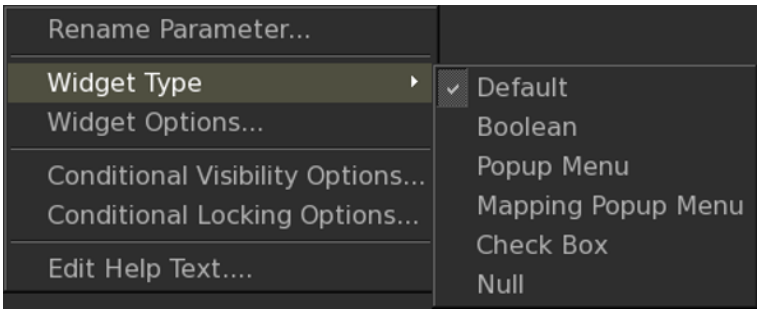
8. Finish editing the Group node's user parameters by clicking > **Finish Editing User Parameters**.
9. Select the group, then click the pop-up user parameter to change between the "sphere", "cube", and "cone" options. The **type** parameter of the PrimitiveCreate node changes to match.

> **Tip:** Katana offers a shortcut to automatically create and populate user parameters. Perform step 3 from the example above, then **Shift**+click the PrimitiveCreate node as well as the group. **Shift**+middle-click and drag from the PrimitiveCreate node's **type** parameter onto the Group node's **Add** menu. A new user parameter of the correct type (in this case a pop-up menu), populated with the applicable entries, is created. You still need to link the new menu to the PrimitiveCreate node's **type** parameter, but this should speed up the process.

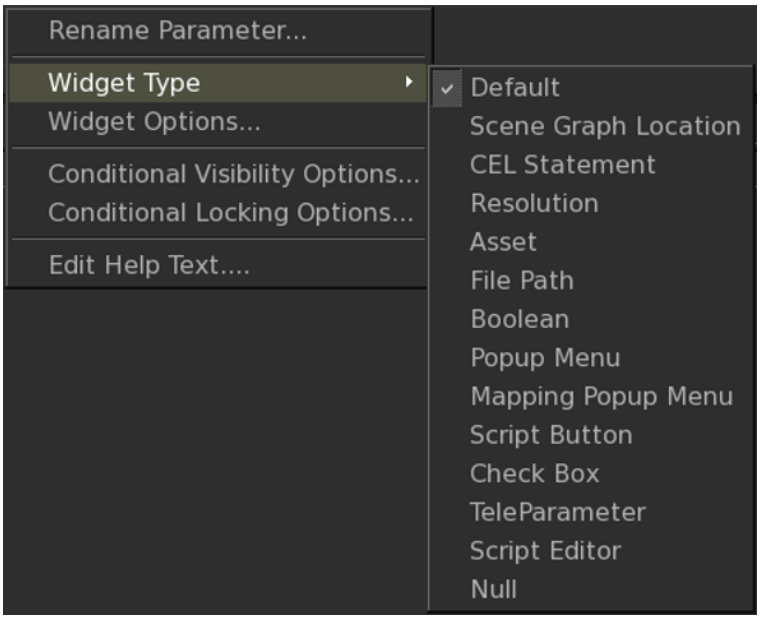
Widget Types

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** shader parameter are shown below.



The widget types for a **String** shader parameter are shown below.



The widget types and widget hint values for the different user parameters are shown in the table below:

Widget Type	Widget Hint Values	Description and Example
Number, String, Button, Toolbar, TeleParameter, and Node Drop Proxy		
Boolean	boolean	Displays two values or options, such as true or false. `<param name="opacity" widget="boolean"/>`

Popup	popup	Displays entries specified in the Widget Options in a dropdown menu.
		`<param name="opacity" widget="popup">`
		`  <hintlist name="options">`
		`  <string value="1.0"/>`

Widget Type	Widget Hint Values	Description and Example
		<string value="1.5"/> <string value="2.0"/> </hintlist> </param>
Mapping Popup Menu	mapper	Similar to **Popup** Menu, but with the option to map values. See Widget Options for more information. <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>
Check Box	checkBox	Similar to Boolean, but displayed as a checkbox. <param name="opacity" widget="checkBox"/>
String, Button, Toolbar, TeleParameter, and Node Drop Proxy		
Scene Graph Location	scenegraphLocation	Widget for specifying locations in the **Scene Graph** tab, for example, **/root/world/geo/pony1** <param name="loc" widget="scenegraphLocation"/>
CEL Statement	cel	Specify a CEL Statement. For more information, see Collections and CEL. <param name="loc" widget="cel"/>
Resolution	resolution	A resolution, for example: 1024x768. <param name="loc" widget="resolution"/>
Asset	assetIdInput	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. <param name="EnvMap" widget="assetIdInput"/>
File Path	fileInput	String parameter representing a file on disk. Uses the

Widget Type	Widget Hint Values	Description and Example

		standard Katana file browser for selection.
		`<param name="texname"`
		`widget="fileInput"/>`

Script Button	scriptButton	A button executing a Python script when clicked.
		`<param scriptText="print 'Hello'"`
		`name="btn"`
		`buttonText="Run Script"`
		`widget="scriptButton"/>`

TeleParameter	teleparam	Creates a parameter that 'teleports' parameters from another source (node, SuperTool, or similar).
		`<param name="EnvMap"`
		`widget="teleparam"/>`

Script Editor	scriptEditor	A field for entering a script as the parameter.
		`<param name="EnvMap"`
		`widget="scriptEditor"/>`

Dynamic Array	dynamicArray	A number or string array of dynamic size. Not available through the UI **wrench** menu.
		`<numberarray_parameter hints="`
		`{';widget': '`
		`dynamicArray'}" 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>`

Multi-line Text	text	Enables a string field to support multiple lines of text. For example, you can set KatanaBlinn.args with the following line: `<param name="BumpMap" widget="text"/>` to set BumpMap to take multiple lines of text and display the expected UI.

Widget Type	Widget Hint Values	Description and Example
String Only		
Attribute Name	attributeName	String parameter value which is the full name of an attribute, with names of ancestor group attributes separated by dots: `xform.translate`
Attribute Type	attributeType	String parameter value which is the name of the attribute's type: `float`
Group Only		
Multi	multi	Creates a group set of parameters within a group.
Number Array Only		
Color	color	Creates a color widget that allows you to set the RGB, HSL, and HSV values.
String Array Only		
Scene Graph Locations	scenegraphLocationArray	Creates three Scene Graph Locations widgets that allow you to set locations.

Note: See **Help** > **Developer Guide** for more on setting hint strings on User Parameters.

Note: See also Adding User Parameters.

Widget Options

Based on the specified widget type, there are a number of options available. In case of a 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.



For example, in the widget options of a **Mapping Popup** menu, if you specify a list of numbers and their labels, they are displayed as a dropdown list.

Note: For more information about user parameters and widget options, refer to Adding User Parameters or for a list of specific widget options for each widget type, refer to Widget Types.

Conditional Behavior

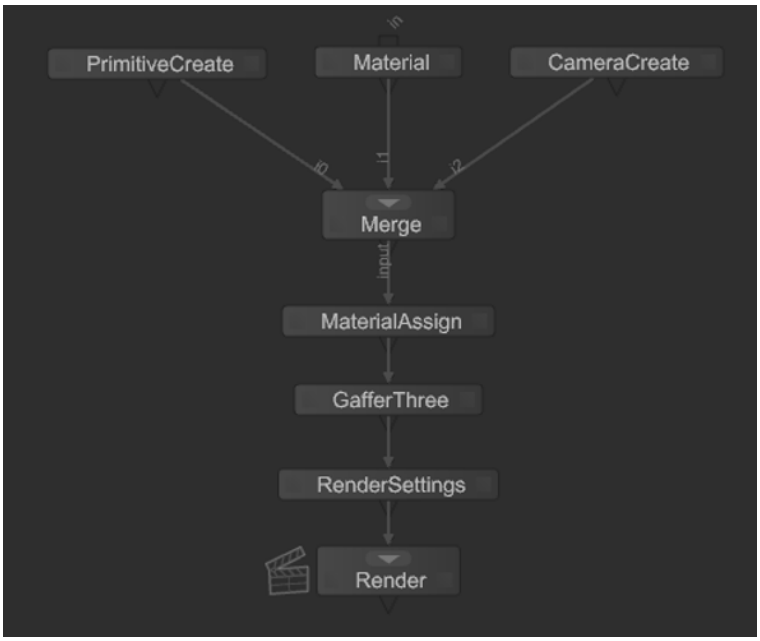
You can make the behavior of user parameters within a macro or group conditional, dependent on any user parameter values. For example, create a group, containing a scene that includes 3Delight and Arnold shaders. Select which shader to use from a pop-up menu, then show and hide shader options using conditional behavior.

Conditional Visibility Example

First, follow the steps below to set up a scene you can use to learn about conditional visibility.

1. Create a Katana scene with a PrimitiveCreate node, a Material node, a CameraCreate node, and a Merge node.
2. Connect the outputs of the PrimitiveCreate, Material, and CameraCreate nodes to inputs on the Merge node.
3. Create a MaterialAssign node, and place it downstream of the Merge node.

4. Add GafferThree and RenderSettings nodes in a series, downstream of the MaterialAssign node. Finally, add a Render node at the end of the chain.



Note: You can overload Material and GafferThree nodes with more than one shader type. For example, a Material node can hold both 3Delight and Arnold shaders.
At render time, only shaders relevant to the selected renderer are considered.

5. In the Material node, add a 3Delight surface shader of type **Material3Delight**, and an Arnold surface shader of type **standard**.
6. In the GafferThree node, add both a 3Delight **spotlight**, and an Arnold **spot_light,** switching profiles to do so.

Note: For more information on how to add lights and assign shaders to them, refer to Getting to Grips with the GafferThree Node.

7. Position the lights.
8. Select all of the nodes except for Render, and press **G** to group all of the selected nodes together. The result is a Group node with a single output, connected to a Render node.

Once you've set up the scene, you can go into the Group node and change the RenderSettings node **renderer** parameter to switch between your available renderers. However, you can also use conditional

visibility to streamline this operation by adding a pop-up menu to the UI of the Group node and linking to the **renderer** parameter on the RenderSettings node by expression.

Follow the steps below to switch between 3Delight and Arnold rendering options using conditional visibility:

1. Select the Group and click > **Edit User Parameters**.
2. Select **Add** > **String** and then select **Widget Type** > **Popup Menu** from the new parameter's menu.
3. Select **Widget Options...** from the new parameter's menu.
4. In the widget options dialog, select **Add** > **New Entry**, so there are two entries in the menu. Edit one entry to read "dl" and the other to read "arnold", then click **OK**.
5. In the Group node's **Parameters** tab, right-click on the pop-up menu widget and select **Copy**.
6. Expand the contents of the Group node in the node graph. Select the RenderSettings node and press **Alt**+**E** to edit the 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 RenderSettings node **renderer** parameter is linked by expression to the selected entry in the Group node's pop-up menu. If you select > **Finish Editing User Parameters** from the group's **Parameters** tab, the pop-up menu now displays as a parameter.

Create New User Parameters for Conditional Visibility

The state of the pop-up menu can also conditionally affect visibility of other user parameters in the Group node. Using the examples scene you've already set up from Conditional Visibility Example, create new user parameters on the group to control the diffuse color values on the 3Delight and Arnold shaders contained within it.

This example also shows how to add conditional behavior so that only the color controls for shaders relevant to the selected renderer are shown:

1. In the Group node's **Parameters** tab, select the > **Edit User Parameters**, then click **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**+middle-click and drag the Material node to the **Parameters** tab to view it.
3. Expand the parameters for the **Material3Delight**, right-click on the **color** parameter (base layer), and select **Paste Expression**.
4. In the Group node's **Parameters** tab, right-click on the second **Color, RGB** user parameter and select **Copy**.
5. Expand the parameters for the Material node's **arnoldSurfaceShader**, right-click on the **kd_color** parameter, and select **Paste Expression**.

The diffuse color values of the 3Delight and Arnold shaders are linked by expression to the color widgets in the group's parameters. If you select **Finish Editing User Parameters** from the group's menu, the color widgets display in the group's **Parameters** tab. Their values affect the diffuse colors of the 3Delight and Arnold shaders.

Hide Conditional Visibility Options

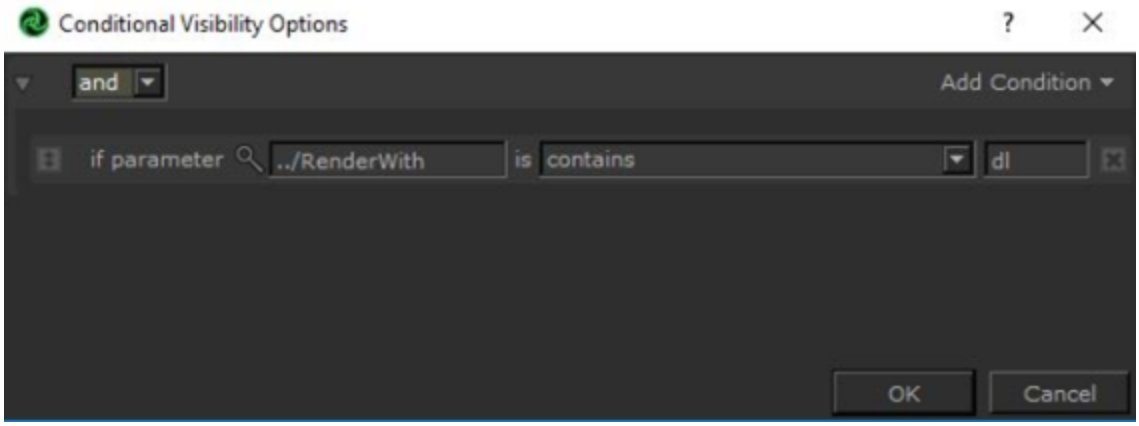
Only one shader at a time is considered, so it would be useful to hide the settings for the shader not applicable to the selected renderer.

1. In the Group node's **Parameters** tab, click > **Edit User Parameters**. Select > **Conditional Visibility Options...** for the first **Color, RGB** widget.

   The **Conditional Visibility Options** dialog opens.

   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** dialog, select **Add Condition** > **contains**. In the text entry field, enter "dl".



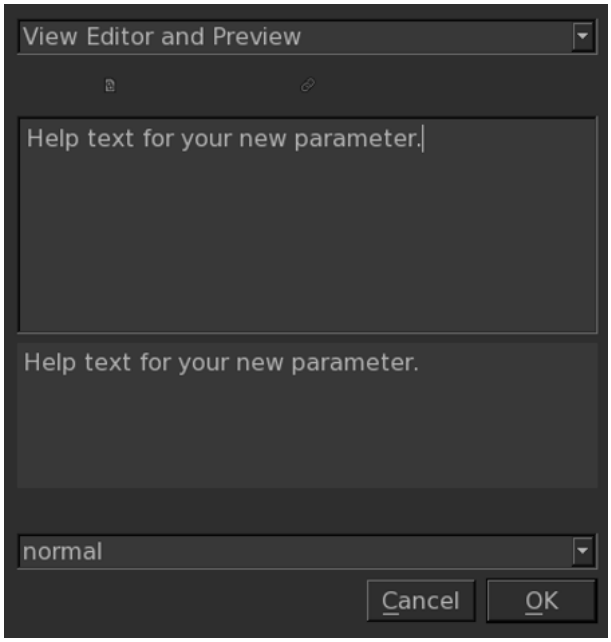
   The condition in this case is **if** <the pop-up menu> **contains** < the string **dl**> then show the target user parameter.
3. In the **Conditional Visibility Options** window, click on the icon to choose the user parameter to test against, and select the pop-up menu from the list.
4. Repeat the process 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 is dependent on the value chosen in the pop-up menu.

Creating Help Text for User Parameters

Within a newly created user parameter, you have the option to create help text.

1. In the new parameter's menu, select **Edit Help Text...** from the options.
2. The **Edit Help Text** dialog is split into two panes: **editor** and **preview**. Using the dropdown menu at the top of the dialog, you can choose to:
   - **View Editor and Preview** - type into the editor, on the top pane, and see the results in the preview pane, on the bottom.
   - **View Only Editor** - type into the editor only, without previewing the results.
   - **View Only Preview** - view the preview window only, without viewing the editor. This is useful for review purposes.



As well as typing into the **editor**, you can insert images and links using the **Insert Image** and **Insert Link** icons above the editor pane.

Note: The image and link options are only available if you have the editor displayed.

3. The dropdown menu at the bottom of the dialog allows you to specify what kind of help text the message is: **normal**, **warning**, or **error**.
4. Click **OK** to save your changes. You can now see your message by clicking on the question mark icon to the left of the parameter.



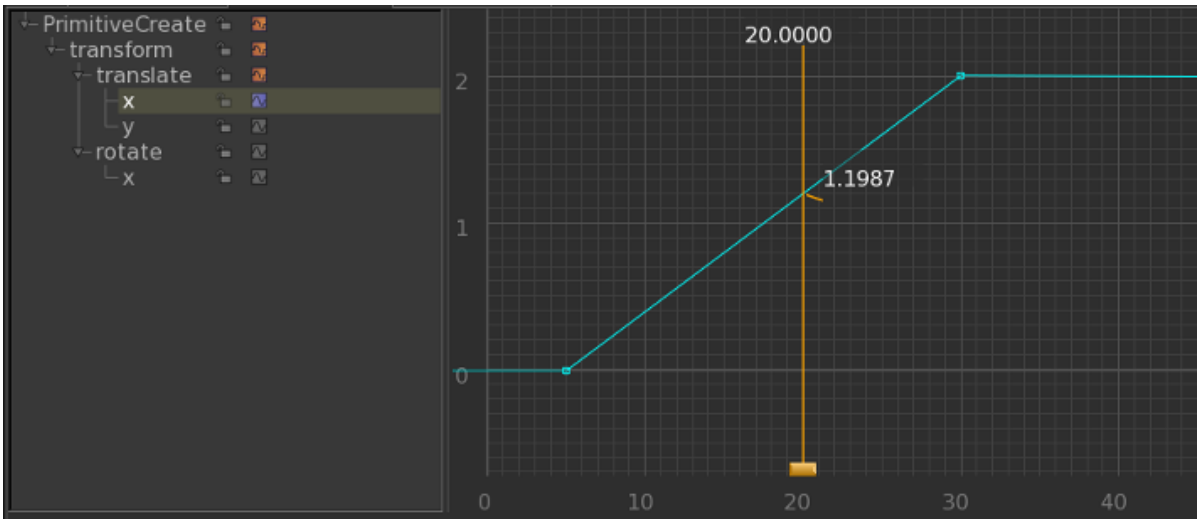
You can also generate help text for a user parameter programmatically instead of setting it through the UI. To do this, set the help text string like the example below:

```
myParameter = NodegraphAPI.GetNode('Group').getParameter
('user.exampleUserParameter')hints = eval(myParameter.getHintString())
hints['help'] = """
This is some example help text<br>
<img src="/workspace/examples/image.jpg"/><br>
<a href="https://www.foundry.com/"> Visit Foundry's website</a>'
"""
myParameter.setHintString(str(hints))
```

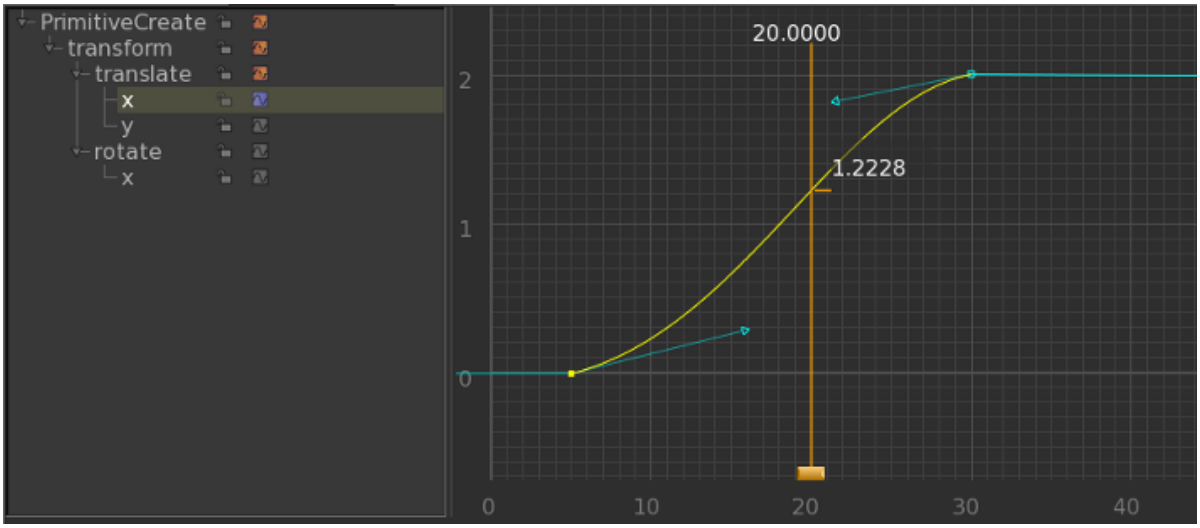
Animation

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.

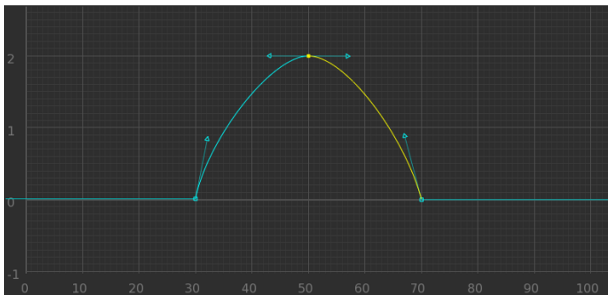
A different animator would then fill in the poses between the keyframes using a technique called tweening, thereby creating the illusion of movement. For some scenes, breakdowns were created to show how the transition from one keyframe flowed to the next. Katana does the animation heavy lifting by interpolating the values between keyframes. You can tell Katana how you want these in-between frames to be generated by specifying a segment function.



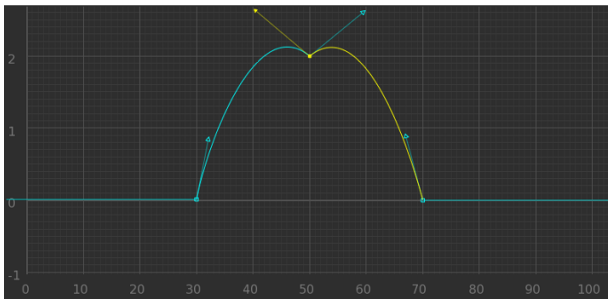
Two keyframes on frames zero and fifty with a linear segment function applied to the first. The most versatile segment function is the **bezier** curve; it uses a mathematical formula to calculate a curve between two anchor points. Bezier curves use four points to interpolate a curve: two anchor points (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.

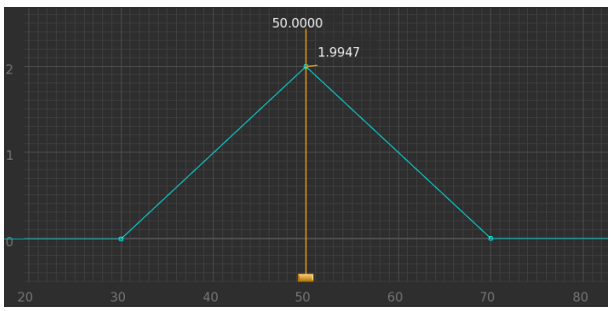


The selected control point handles, shown in yellow, form a tangent around the keyframe.

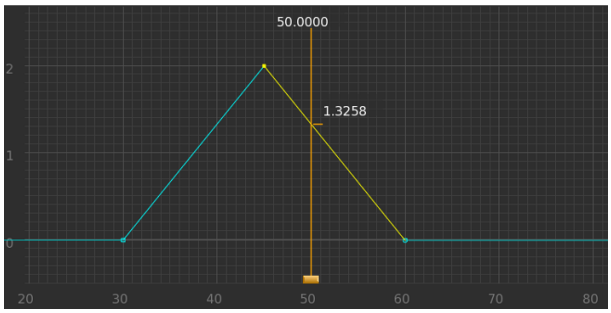


Here, a straight line between control points would not pass through the keyframe; hence the tangent is broken.

Breakdowns within the **Curve Editor** maintain the relative time between the keyframe before and the keyframe after.



Keyframes have been placed on frames 30, 50, and 70. The middle keyframe, on frame 50, has been converted into a breakdown.



Moving the third keyframe from frame 70 to frame 60, automatically moves the breakdown to frame 45.

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 and Baking a Segment of the Curve for more information.

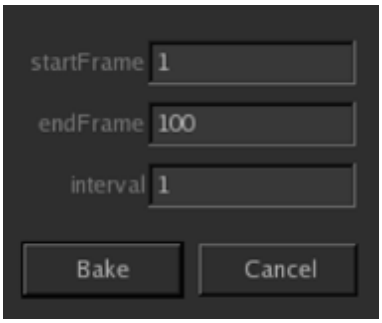
Baking a Curve

Whether from an expression or a keyframed curve, you can convert part or all of a curve to keyframes.

Generating Keyframes from a Curve or Expression

1. Right-click on the parameter.
2. Select **Bake to FCurve...** .

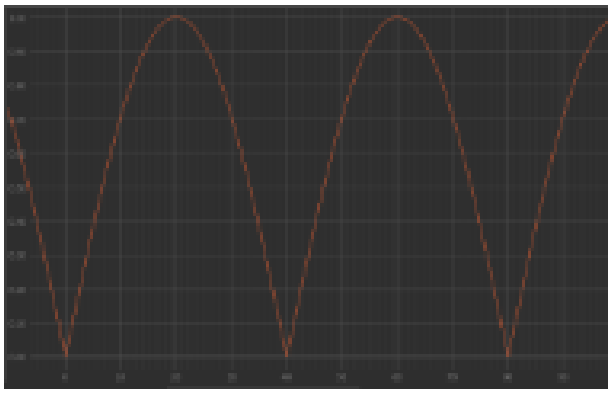
   The **Bake to Curve** dialog displays.



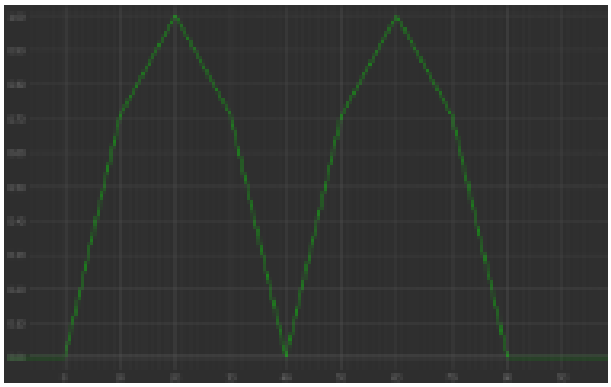
3. Change the dialog values to suit the curve you are creating. You can change the:
   - **startFrame** - the frame to start generating keys.
   - **endFrame** - the last frame to generate a key.
   - **interval** - how often to generate a key (in frames).
4. Click **Bake**.

   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 baked curve 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.

To export a curve:

1. Right-click on the parameter to export.
2. Select **Export FCurve...** .

To import 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:

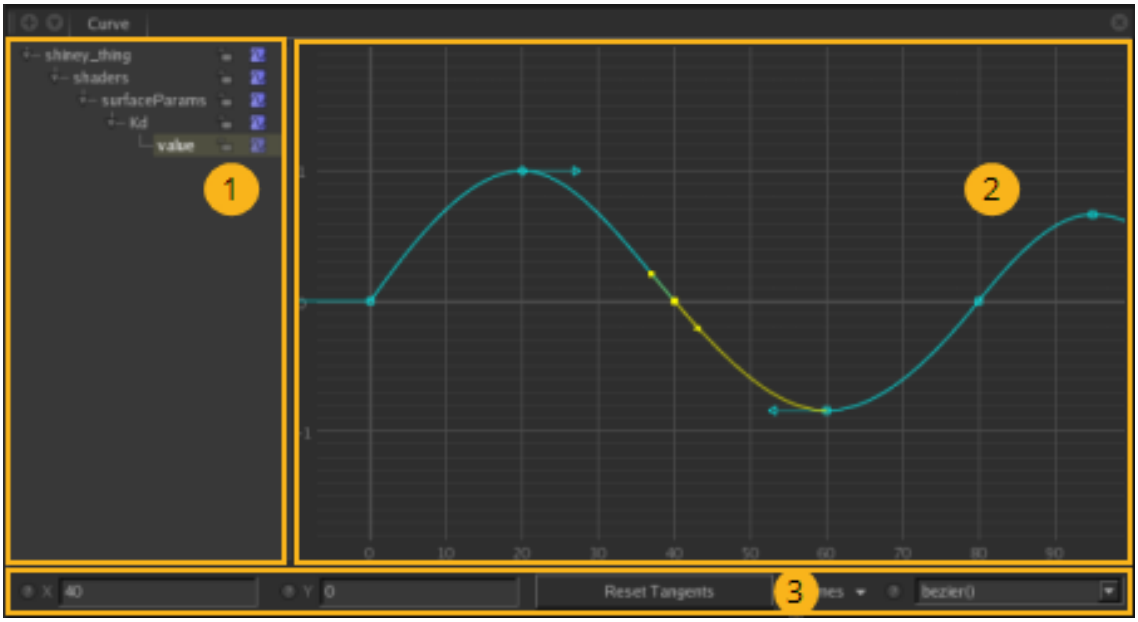
1. Right-click on the parameter.
2. Select the **Show Curve** menu item.

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 or Hiding a Curve

You can lock a parameter to stop its curve from being editable within the **Curve Editor**.

To locking a parameter and stop it from being editable within the Curve Editor, click  within the hierarchical view in the **Curve Editor**. If you then want to unlock the parameter again, click  .



Note: Parameters that are expressions are always locked and cannot be modified within the **Curve Editor**.

Even though a parameter has **Show Curve** selected, you may not want to display it within the **Curve Editor** graph.

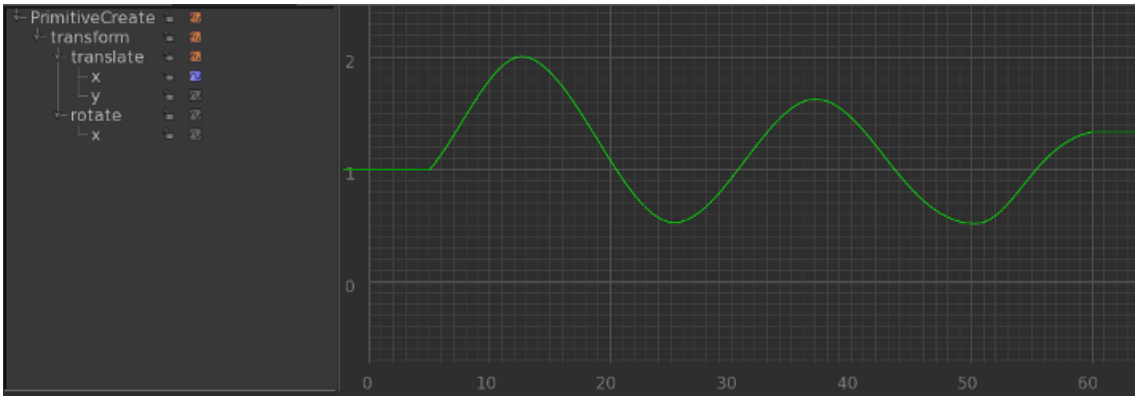
To hide a parameter curve within the Curve Editor, click  within the hierarchical view in the **Curve Editor**.

If you then want to show the parameter curve again, click  within the hierarchical view in the **Curve Editor**.

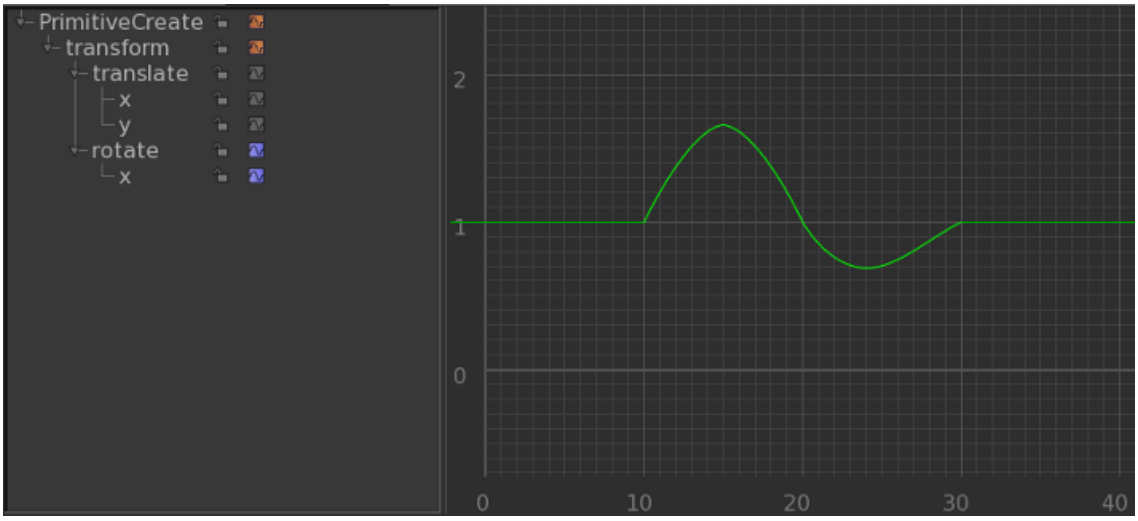
Switching the Display of a Parameter's Children

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

You can set keys quickly and easily within the **Curve Editor** with the 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 and Moving Keyframes

You can selecting keyframes by clicking on them or by marquee dragging over them. If you want to select all keyframes for a curve, double click the curve. To add a keyframe to the current selection, hold **Shift** while selecting the keyframe(s). If you want to remove that keyframe again, or any keyframe in the selection, hold **Ctrl** while selecting the keyframe(s).

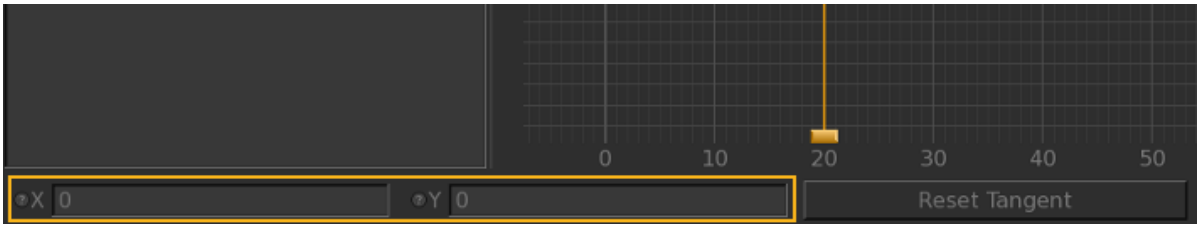
You have two ways to move keyframes within the Curve Editor: using the mouse or using the **X** and **Y** input fields.

To move keyframes using the mouse:

1. Select the keyframe(s) you want to move.
2. Click-and-drag one of the selected keyframes.

To move 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.

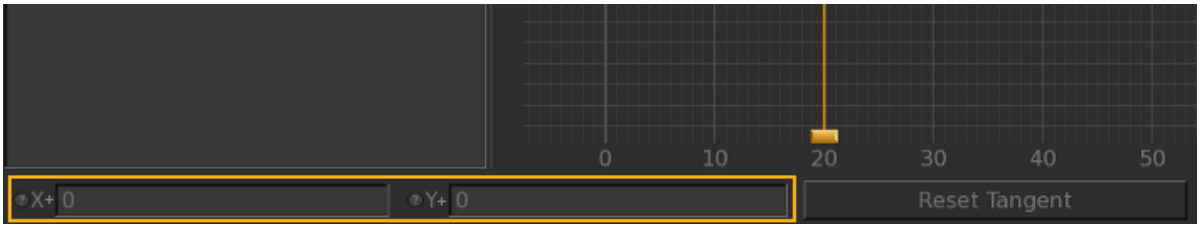


The values entered into **X** and **Y** are absolute and not relative. For instance, entering 10 in the **X** input field, moves the keyframe to frame 10.

To move 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 and Display Elements

Katana provides a number of ways to change the frame range and parameter value range in the Curve Editor graph.

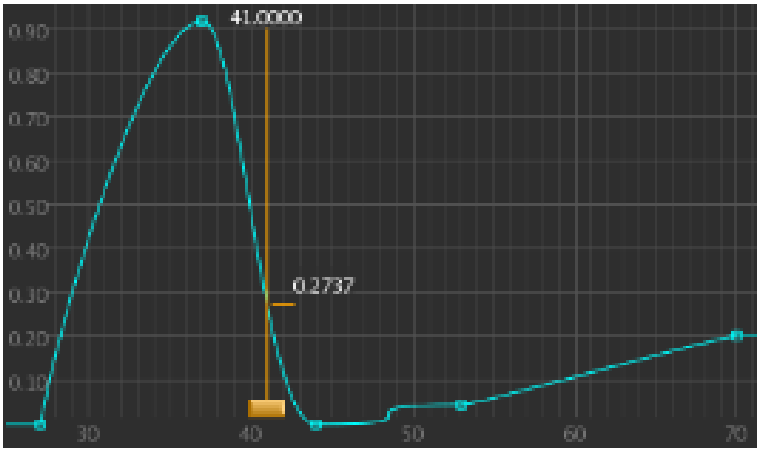
Action	Description
Panning	
Panning in the graph	Middle-click and drag within the graph area.
Panning in a single axis	**Shift**+middle-click and drag within the graph area.
Zooming	
Zooming in or out	Use the scroll wheel to scroll up (zoom in) and down (zoom out). Alternatively, press the **+** (Plus) key to zoom in or press the **-** (Minus) key to zoom out.
Framing	
Framing all keyframes in the graph	Right-click and select **Frame** > **All** > **Frame All** (or press **A**).
Framing all keyframes in the graph in the X axis	Right-click and select **Frame** > **All** > **Frame All X Only**.
Framing all keyframes in the 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**). Alternatively, right-click and select **Frame** > **Selected** > **Frame Selected**.
Framing the selected	Right-click and select **Frame** > **Selected** > **Frame Selected X Only**.

Action	Description
keyframes in the X axis	
Framing the selected keyframes in the Y axis	Right-click and select **Frame** > **Selected** > **Frame Selected Y Only**.

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.

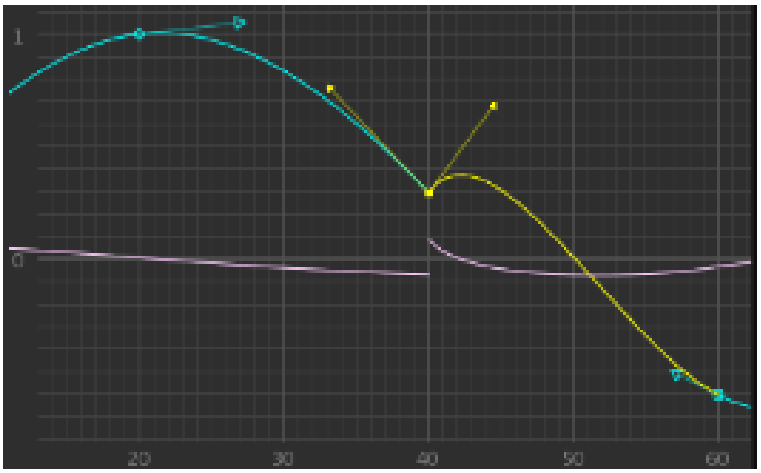
Displaying the Domain Slider

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.

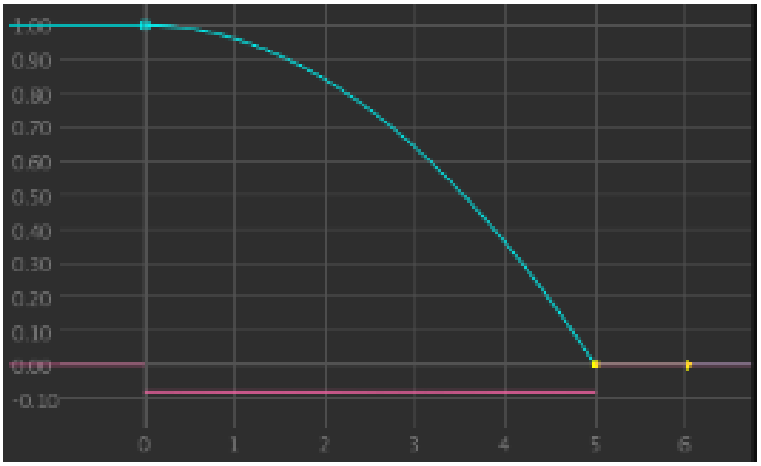
To toggle the display of a curve's velocity:

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:

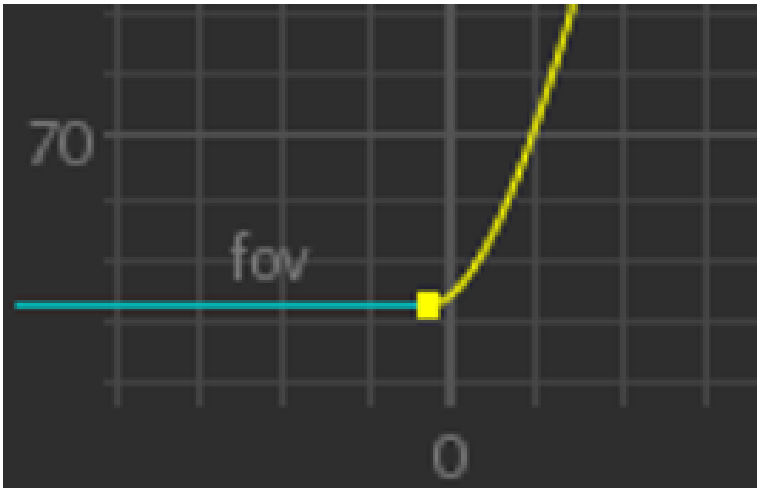
1. Select the curve(s) within the hierarchical view.

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.

You can snap the frame number of a keyframe while moving it in the **Curve Editor** tab 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.

To snap only a keyframe's value while moving it in the **Curve Editor** tab, right-click and select **Grid Snapping > Y Snap to Grid**. Katana snaps value changes to the horizontal grid lines.

If you want to turn off keyframe snapping altogether,

- 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.**

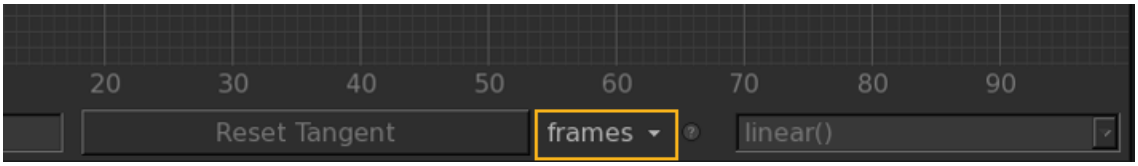
  Katana no longer snaps keyframe changes in the y axis.

- 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 pre-defined 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.)



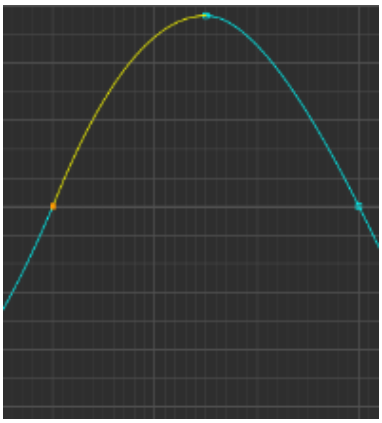
Tip: To cycling through the preset snapping options (**Off**, **Frames**, **Grid**, and **Custom**), right-click and select **Grid Snapping** > **Cycle Snapping** (or press **S**).

Locking and Deleting Keyframes

To locking keyframes in order 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.

To unlock keyframes again:

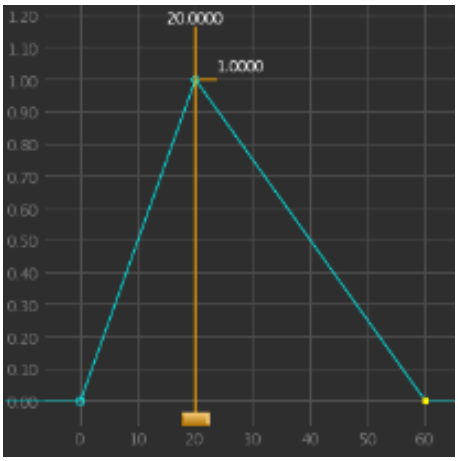
1. Select the keyframes to unlock.
2. Right-click and select **Keyframe** > **Unlock**.
   Katana unlocks the keyframes and turns them yellow.

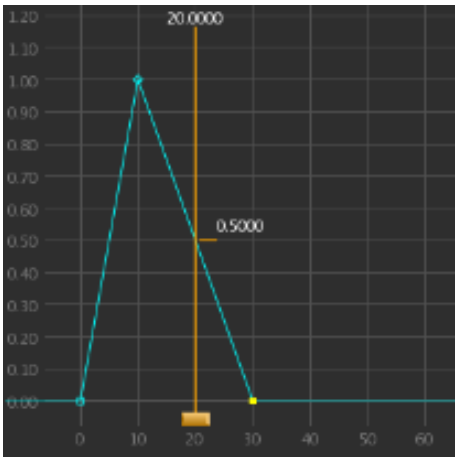
If you want to delete keyframes:

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.

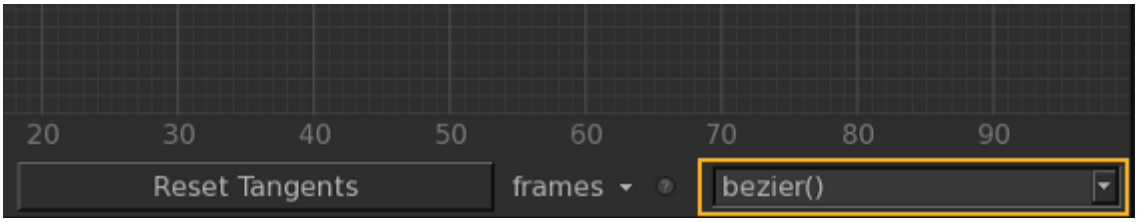
Segment Functions

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()**.

To change 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**.

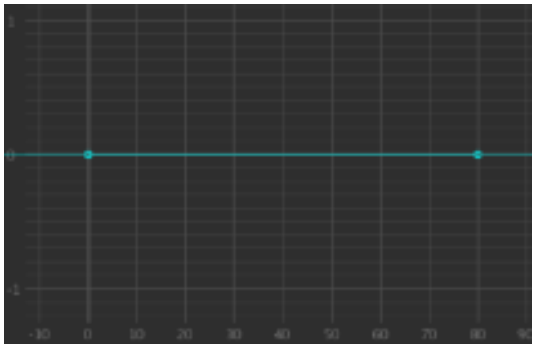


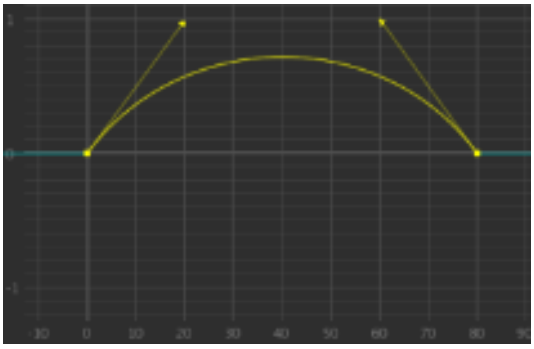
Available Segment Functions

The following are a list of available segment functions:

- **bezier()**

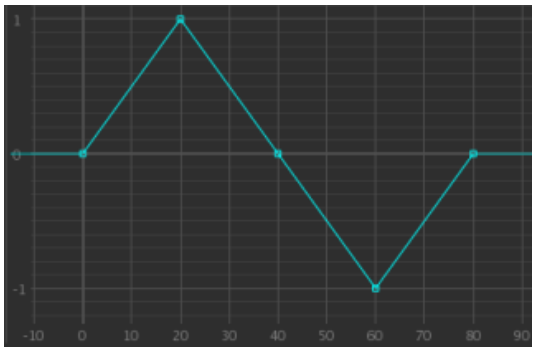
  The bezier segment function is the most versatile. It uses four points - the keyframes at the start and end, and two control points - to define the segment. The control point position is shown with an arrowhead. The weight of a control point, which determines how strong its influence is over the generated curve, is determined by the length of the handle.

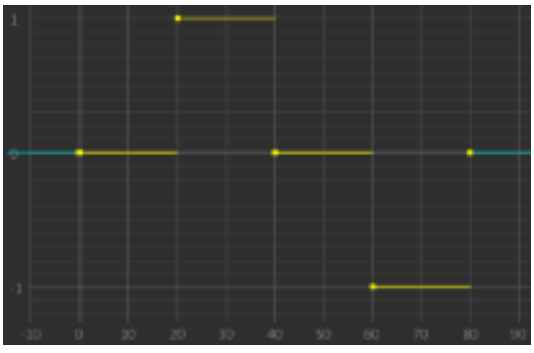




- **constant()**

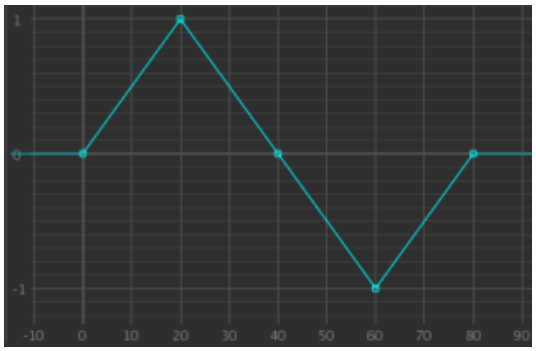
  The constant segment function uses the keyframe's value for the entire segment.

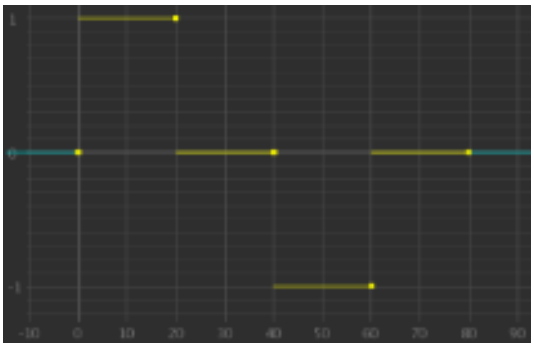




- **constant_next()**

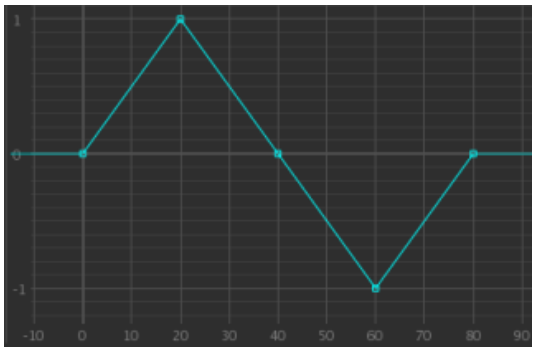
  The constant_next segment function uses the next keyframe's value for the entire segment.

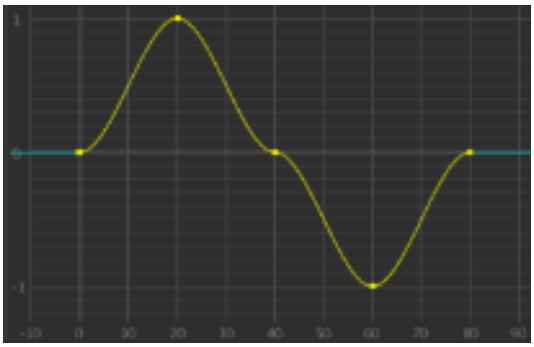




- **ease()**

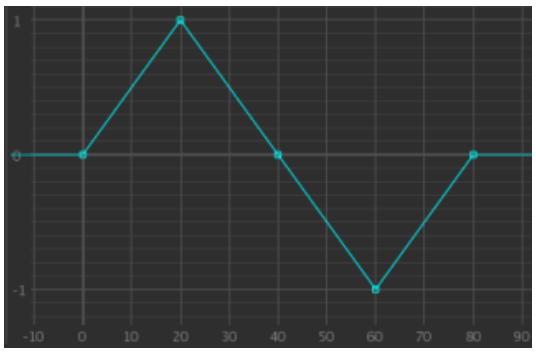
  The ease segment function flattens out the segment at its beginning and end. This is similar to having flat tangents on the two control points when using bezier curves.

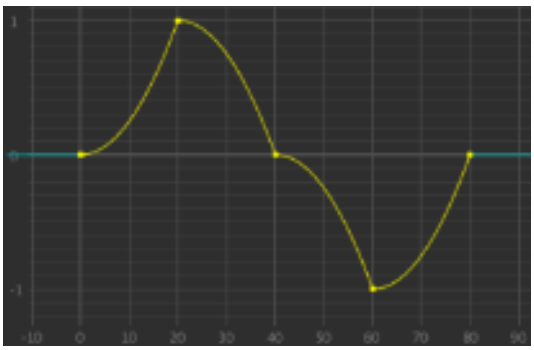




- **easein()**

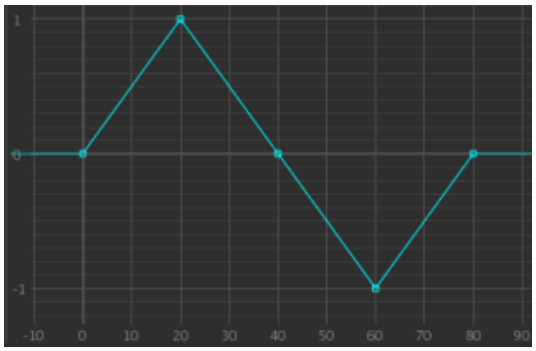
  The easein segment function starts the segment flat and then maintains the same acceleration until it reaches the next keyframe. This results in the velocity curve for the segment being a straight line that starts at zero.

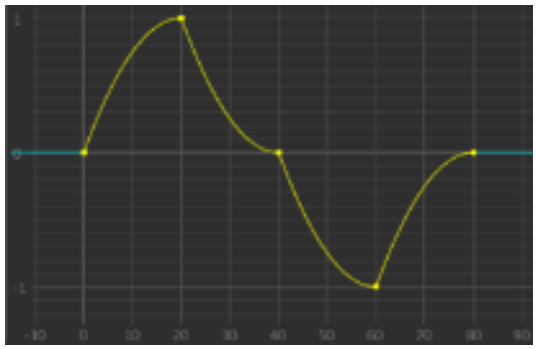




- **easeout()**

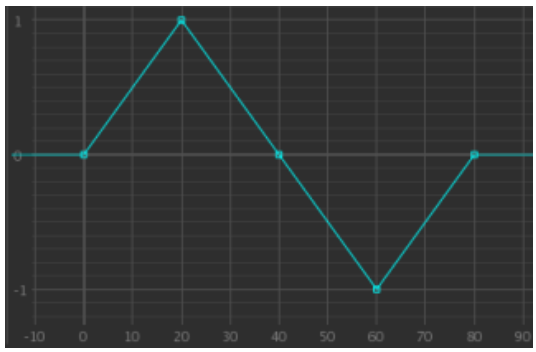
  The easeout segment function finishes the segment flat while maintaining a constant acceleration throughout the segment. This results in both the velocity curve for the segment being a straight line that ends at zero.





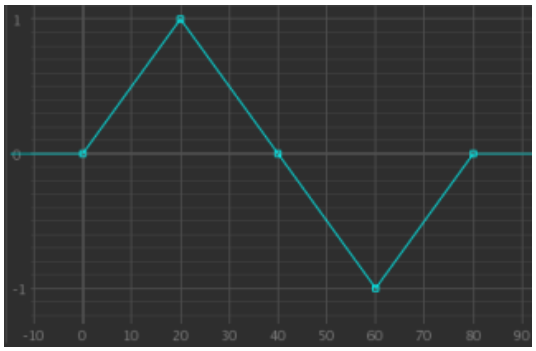
- **linear()**

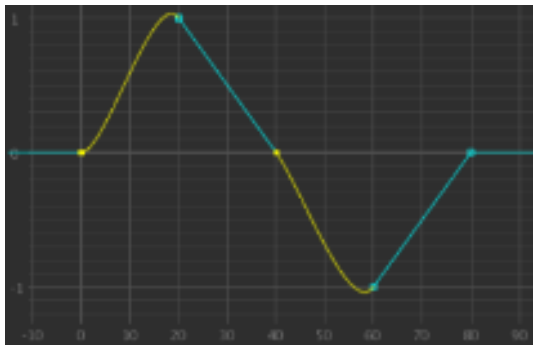
The default segment function. The values from one keyframe move in a straight line to the next keyframe.



- **match()**

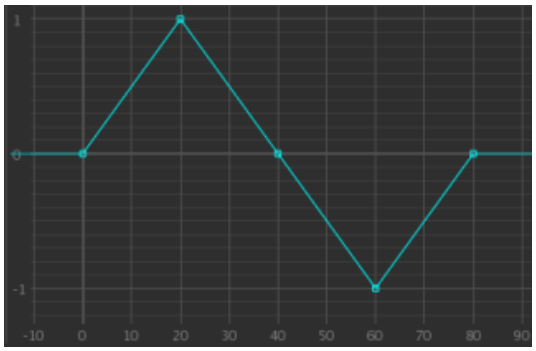
The match segment function gives the segment the same velocity (rate of change) at both the start and end of the segment.

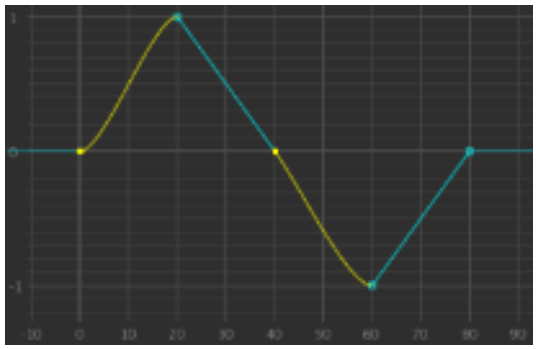




- **matchin()**

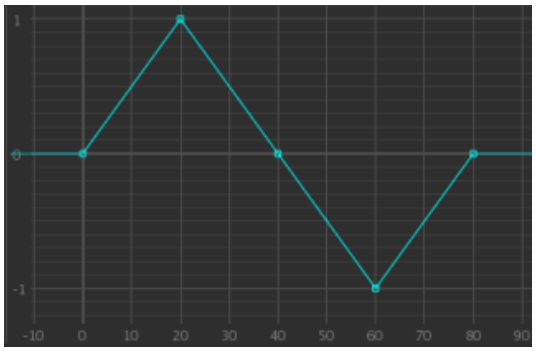
A segment with the matchin segment function begins with a velocity that matches that at the end of the previous segment, the segment ends with zero velocity. This has the effect of making the tangent at the start match the slope of the previous segment and the tangent at the end flat.

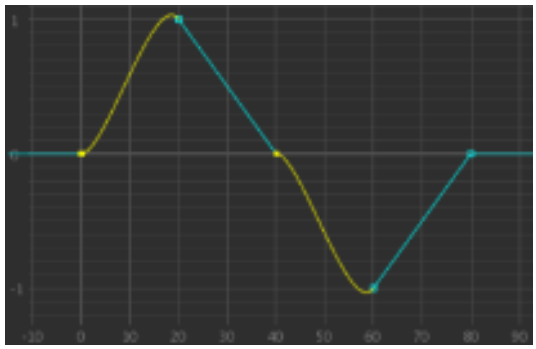




- **matchout()**

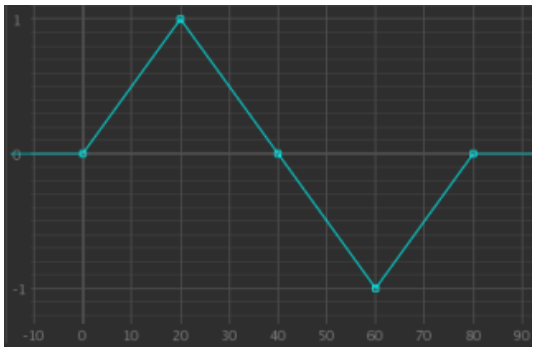
  A segment with the matchout segment function begins with zero velocity and ends with a velocity that matches that at the beginning of the next segment. This has the effect of making the tangent at the start flat and the tangent at the end match the slope of the next segment.

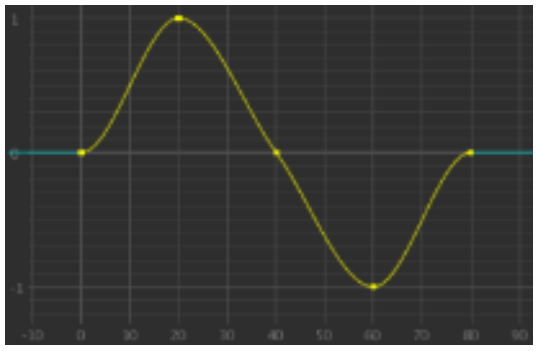




- **spline()**

  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.



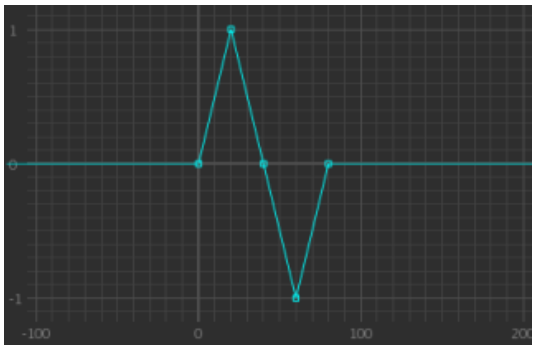


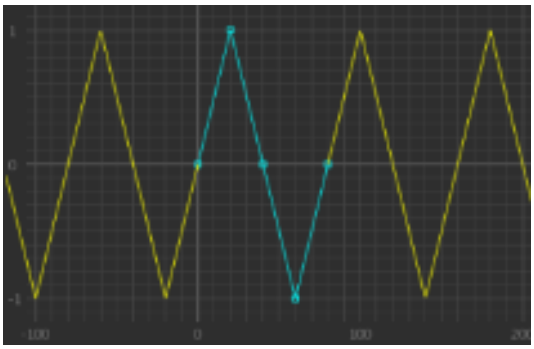
Available Extrapolation Functions

Extrapolation functions are used to extend the behavior of a curve before the first keyframe and after the final keyframe. The available options are:

- **cycle()**

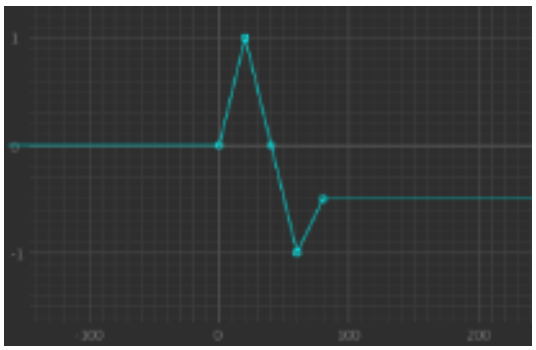
The cycle extrapolation function repeats the curve an infinite number of times either before (if applied to the segment before the first keyframe) or after (if applied to the segment after the last keyframe).

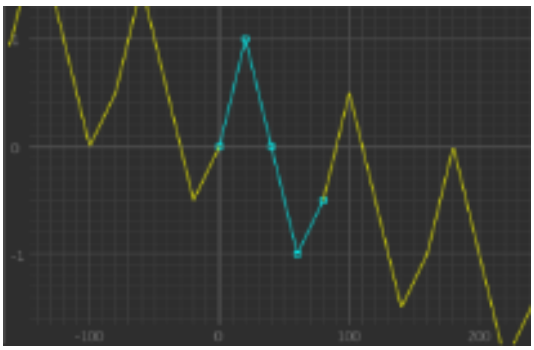




- **cycle_offset()**

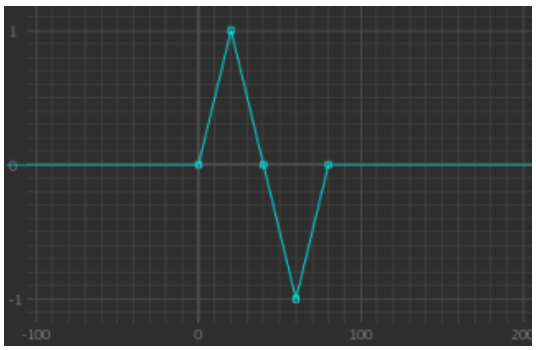
The cycle_offset segment function only works on the segments at the start or end of a curve. It should not be used on a keyframe. It repeats the curve an infinite number of times; each time the curve repeats the new beginning keyframe starts from the end keyframe from the previous cycle, thus offsetting the curve.

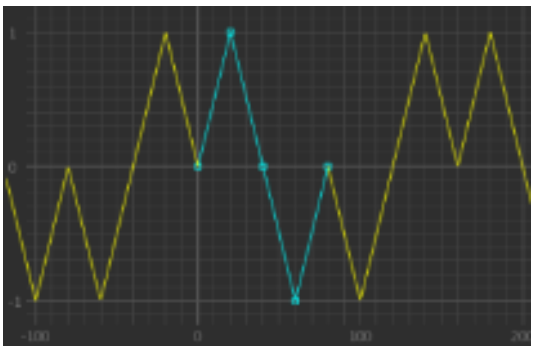




- **mirror()**

The mirror segment function only works on the segments at the start and end of a curve. It continuously flips the curve vertically.



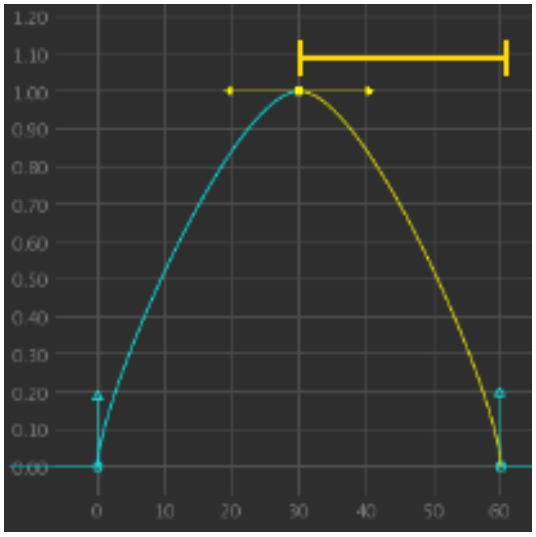


Tip: It is also possible for you to type your own segment or extrapolation function in the dropdown menu. The function must use Python syntax:
x( ) can be used to represent the current frame. For instance, sin(x( )*pi/20).

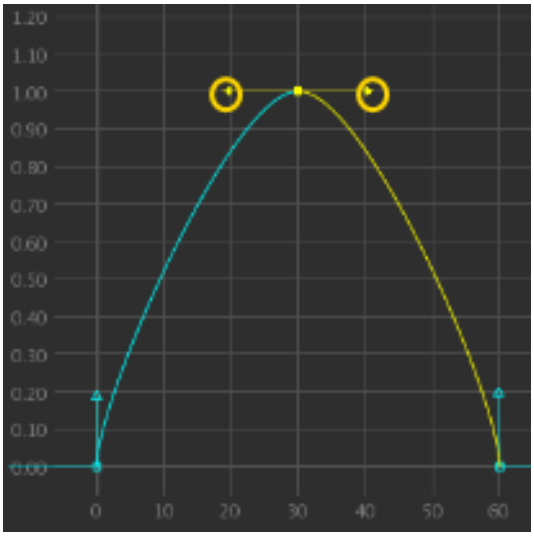
Changing the Control Points of a Bezier Segment Function

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.

To change the tangent type at a keyframe:

1. Select the keyframe(s) to change the control points.
2. Right-click and select **Tangent** > **Type** > **...** .

To 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

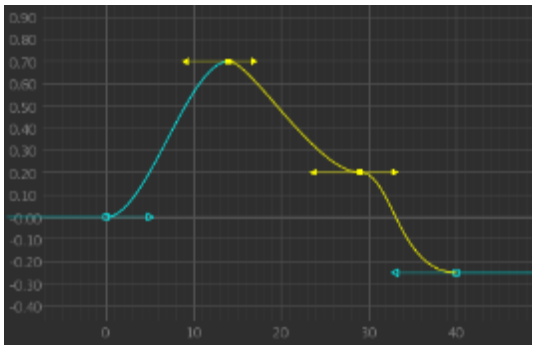
The following are a list of tangent types:

- **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.

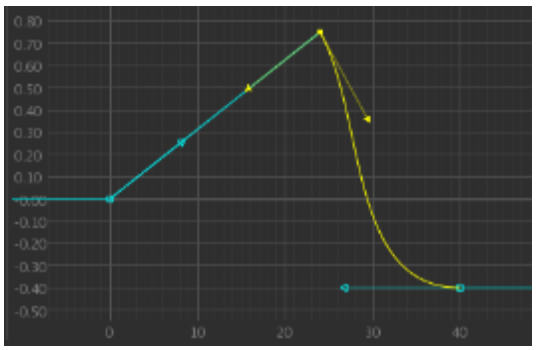
- **Flat**

  The Flat tangent type makes the control points sit horizontally either side of the keyframe. All the keyframes are using the Flat tangent type.



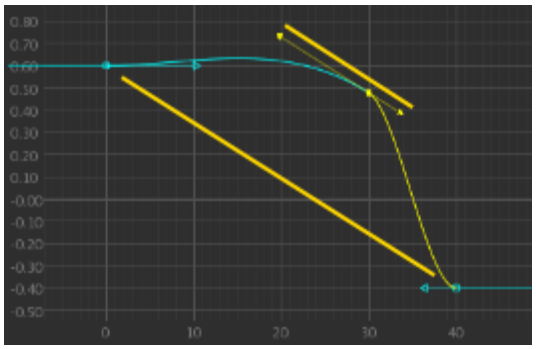
- **Linear**

  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.



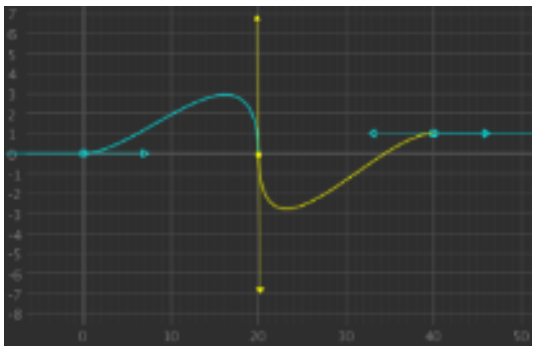
- **Smooth**

  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.

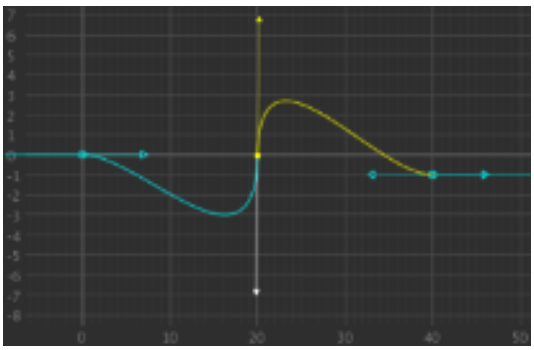


- **Smooth Normal**

  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.



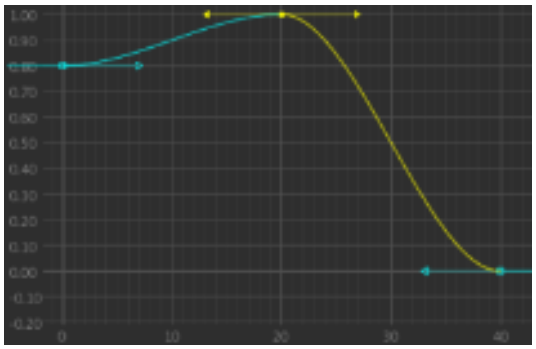
With the right keyframe above the left, the curve goes down through the middle keyframe.



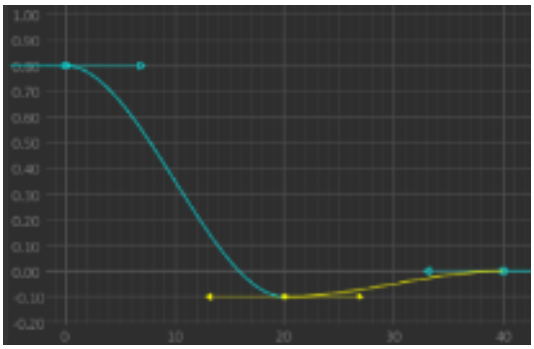
With the right keyframe below the left, the curve goes up through the middle keyframe.

- **Plateau**

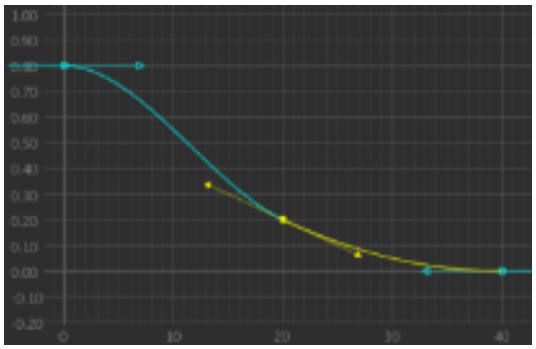
  The Plateau tangent type uses the Flat and Smooth tangent types depending on its keyframes location relative to the keyframes on either side. If the keyframes on either side are both above or both below the tangent's keyframe, then the Flat tangent type is used. If the tangent's keyframe falls between 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.



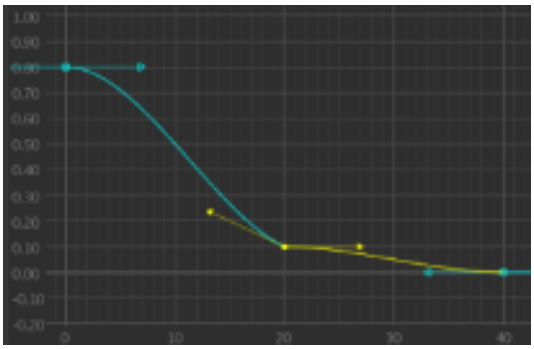
Here the Plateau tangent type uses the same algorithm as the Flat tangent type.



Once again the Flat tangent type is used.



Here the Plateau tangent type uses the same algorithm as the Smooth tangent type.



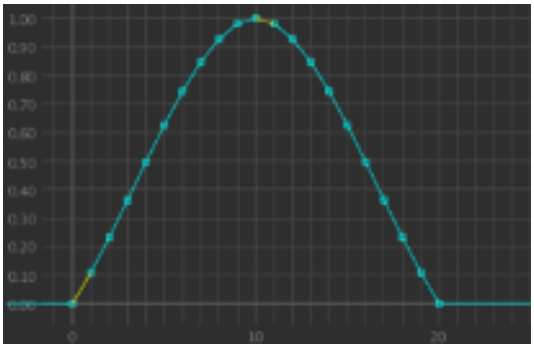
As the lower control point would drop below the keyframe to the right, that control point becomes Flat.

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.

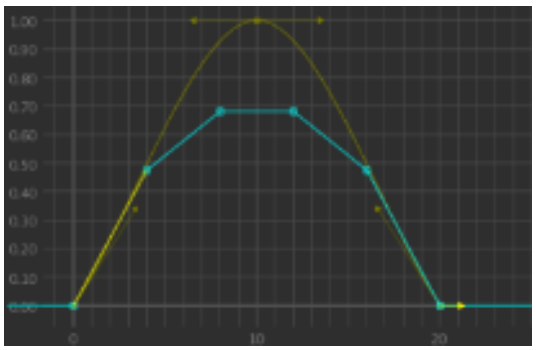
To smooth a segment of the curve:

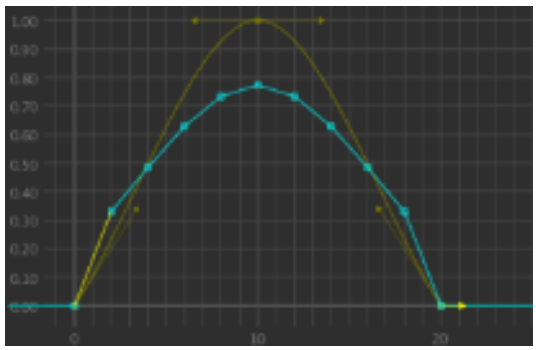
1. Select the keyframe at the start of the segment you want to smooth.
2. Right-click and select **Transform** > **Smooth...** .
   The **Smooth** dialog displays.



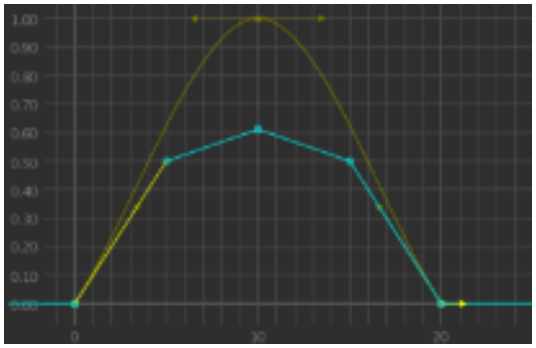
3. Change the values within the dialog where appropriate:
   - **Step Size** - how often to create a keyframe.
   - **Radius** - how much to smooth the curve (higher values for smoother, lower values for closer to original).
   - **Filter** - which algorithm to use, **Triangle** or **Box**.
4. Click **Apply** to smooth the curve.

Note: For the best results, smooth multiple segments at once by selecting a number of keyframes together.



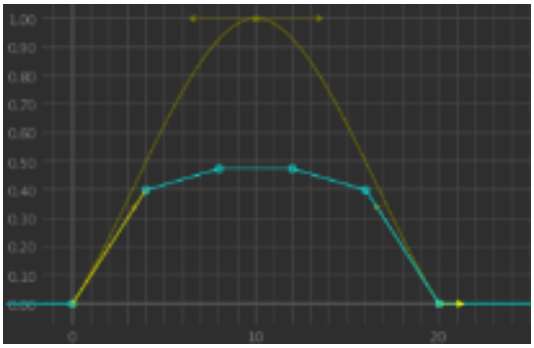


Smoothing with the default settings: **Step Size** 4, **Radius** 2, and Triangle **Filter** (the original curve is ghosted out).



Step Size 5 and **Radius** 2.

Smoothing with **Step Size** 2 and **Radius** 4.



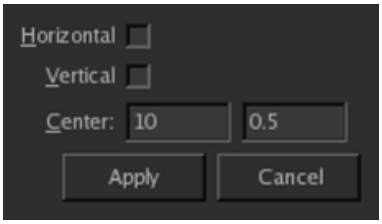
Step Size 4 and **Radius** 4.

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.



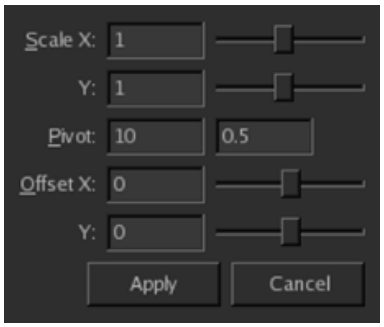
3. Select whether you want to flip the curve horizontally or vertically or both:
   - **Horizontal** (press **Alt**+**H**) - flips the curve horizontally.
   - **Vertical** (press **Alt**+**V**) - flips the curve vertically.
   - **Center** (press **Alt**+**C**) - the point at which to flip the curve (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:

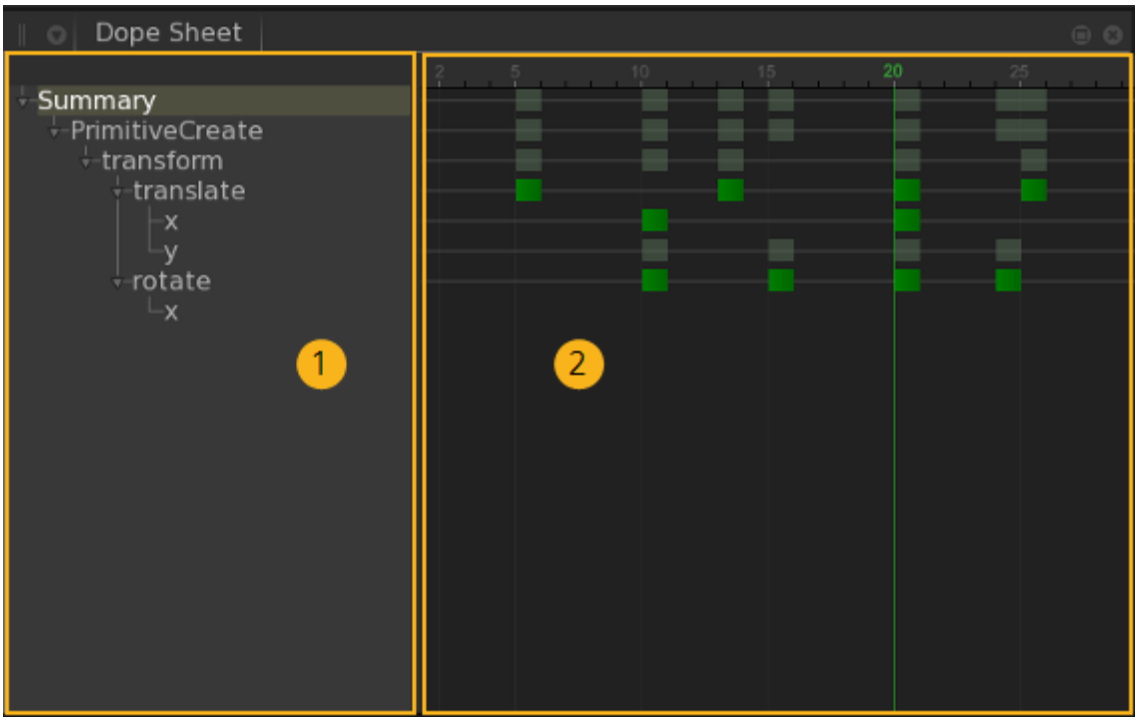
1. Select the curve or a curve's keyframe.
2. Right-click and select **Transform** > **Scale & Offset...** .
   The **Scale & Offset** dialog displays.



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.



The numbered list that follows corresponds to those numbers in the image above:

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 the **+** (**Plus**) key to zoom in or the **-** (**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**.

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.

To select a keyframe, click on it or drag a marquee around it. To select multiple keyframes, drag a marquee around all the keyframes you want to select, or right-click and choose **Select All** from the menu (or press **Ctrl**+**A**) to select all visible keyframes.

You can add to a selection at any time by clicking, or dragging a marquee over the keyframe(s), while holding the **Shift** key. If you want to remove from a selection, click it, or drag a marquee over the keyframe(s) while holding the **Ctrl** key.

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.

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.

1. Select the keyframe(s) to copy.
2. Right-click and select **Copy Selected Key**(**s**) (or press **Ctrl**+**C**).
3. 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.

   Alternatively, if you are using the **Timeline**, move the current frame to where you want to insert the new keyframe(s) and press **Ctrl**+**V**.

Deleting Keyframes

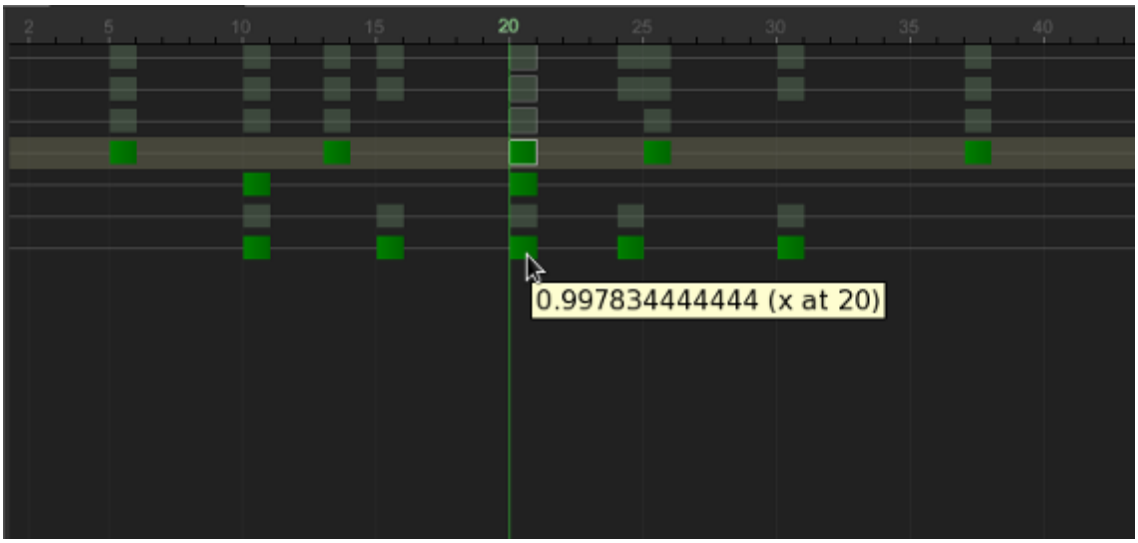
To delete keyframe(s):

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.



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.

Using the Scene Explorer and Scene Graph

In Katana, there are two panels which you can use to view and manage the hierarchical structure of a scene.

- **Scene Explorer**
- **Scene Graph**

The **Scene Explorer** is a newer and more extended version of the **Scene Graph**, with a versatile layout and additional tools. While it was designed primarily to help manage scenes with USD, its options will suit any workflow - USD, geolib (Katana's legacy data) only, or a combination of the two. In particular, the side-by-side toggle view between USD and Katana geolib can help manage both simultaneously.

See Using the Scene Explorer.

The older **Scene Graph** does not support USD, but it still has all the other functions you will need for working with geolib scenes if you prefer to use this.

See Using the Scene Graph.



Note: Any functions of the Scene Graph also exist in the Scene Explorer.

Using the Scene Explorer

We introduced the Scene Explorer tab as part of our growing support for USD, making navigating and exploring scenes easier. The scene explorer contains panels for two scene graphs:

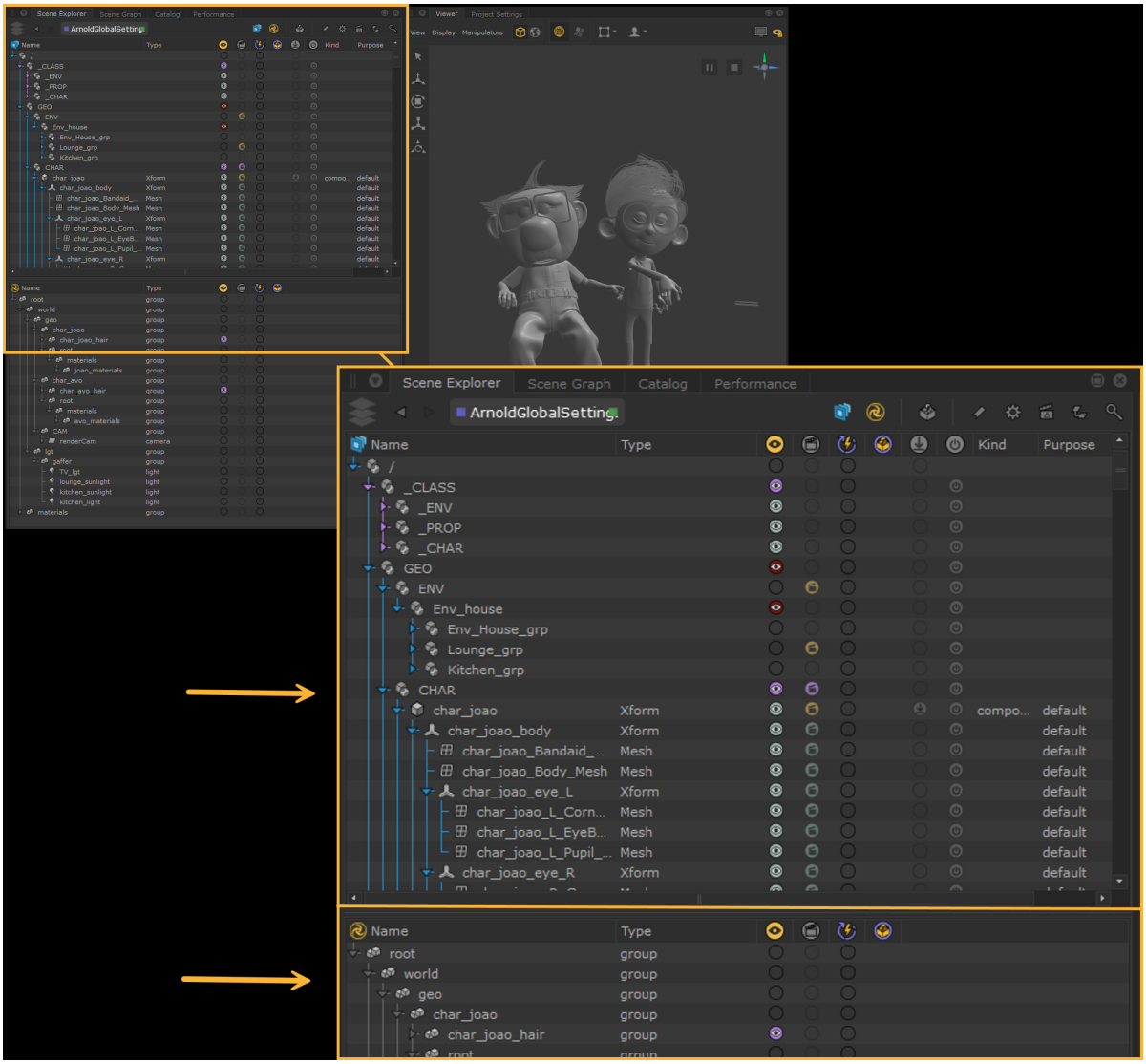
- **USD Scene Graph**- Inspect and manipulate the properties and attributes of prims in the stage hierarchy.
- **Katana Scene Graph** - The Katana Scene Graph for non-USD items.

There is also a panel above the scene graphs, showing the **Collections** in the project. This is collapsed by default. See more about this at Collections in the Scene Explorer.

Note: The Katana Scene Graph is explained in Using the Scene Graph.

The workflow for the Scene Explorer will be familiar if you've used the Katana Scene Graph. Both the USD Scene Graph and the Katana Scene Graph panels in the Scene Explorer share working set columns for essential operations, including visibility in the viewer and renderer. However, the USD Scene Graph adds two USD-specific working set columns so that you can have more control over payloads and prim activation.

As you can see in the image below, the **Scene Explorer** tab includes both the USD Scene Graph and the Katana Scene Graph beneath.



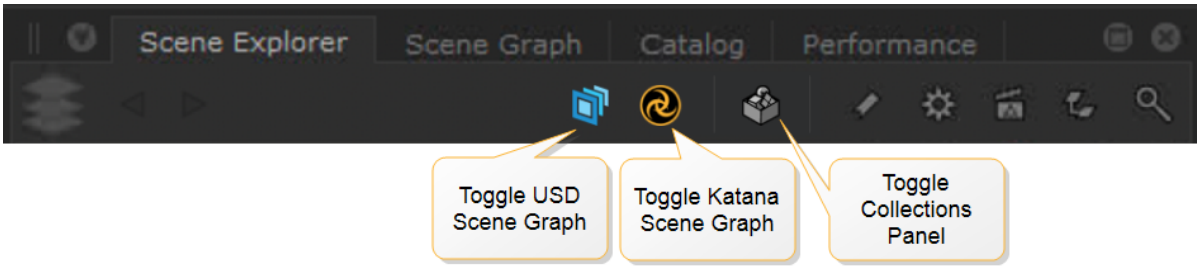
The Scene Explorer. The top section is the USD Scene Graph, and the bottom is the Katana Scene Graph.

Each scene graph is identified by the USD or Katana logo in the top-left corner of the panel next to the **Name** column.

Note: To hide the Katana Scene Graph in this panel, either drag the top of the panel bar either to the top or bottom or by toggling either the USD or Katana logo located at the top of the panel.

Viewing Scene Graphs

Use the three selection buttons at the top of the Scene Explorer tab to display or hide the USD Scene Graph, or/and the Katana Scene Graph, or/and the Collections panel.

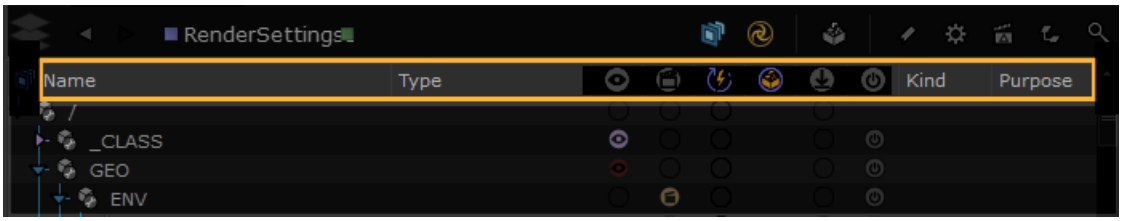


If a panel is hidden, its icon is greyed-out.

USD Scene Graph Columns

Data Columns

The USD Scene Graph part of the Scene Explorer tab has four columns of key information for each item:



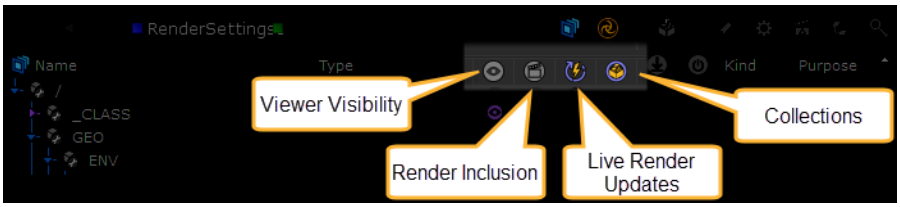
- **Name** - The name of the prim.
- **Type** - The data category of an element.
- **Kind** - The organizational role of an element in the hierarchy. It is categorized as values such as **component** for leaf level items or **assembly** for a collection of components. This can be modified using the UsdKindSet node in Katana.
  See Kind in Pixar's USD Glossary.
- **Purpose** - The category for visibility of components or primitives. For example, a **default** purpose generally means that it is a primary element of the scene, and is involved in various processes, such as rendering, simulation, and interaction within the viewport. A **proxy** purpose would indicate its use in viewports but not in renders, whereas a **render** purpose indicates use in renders but not viewports.
  See Purpose in OpenUSD's USD Glossary.

By default all purpose types are displayed in the Viewer. However, you can choose which are shown by selecting **Display** > **Purposes** in the **Viewer** tab.

Note: When prims with the proxy purpose are added to Katana, by default they have their render visibility turned off.

Common Working Set Columns

The USD Scene Graph shares four working set columns with the Katana Scene Graph:

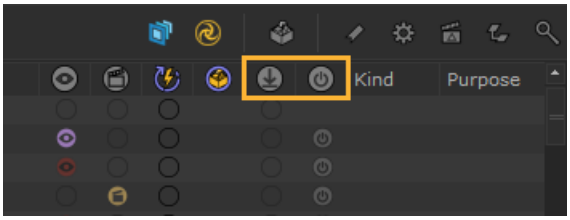


The Viewer Visibility, Render Inclusion, and Live Render Updates columns control visibility in the viewer and renderer. To find out how to work with these, see Working Sets.

The Collections working set shows which locations have a collection on them. See Collections in the Scene Explorer.

USD-Specific Working Set Columns

The USD Scene Graph includes two columns for payload management and prim activation/deactivation. See Payloads in the Scene Explorer, and Activating and Deactivating Prims.



Styling in the Scene Explorer

Prim Specifiers determine the order of operations when building up a USD scene. Seeing whether prims are **defined**, **over** or **class** helps you to quickly debug your USD composition.

The Scene Explorer allows you to quickly gain visibility of your scene setups, via visual cues to indicate information such as prim specifiers, visibility, instance proxies..

In the Scene Explorer, the tree lines **styling** will change based on certain prim rules:

- **Defined** prims show a **blue branch** in regular font
- **Abstract (class)** prims show a **purple branch**
- **Instance proxy** prims are displayed as **dotted lines**

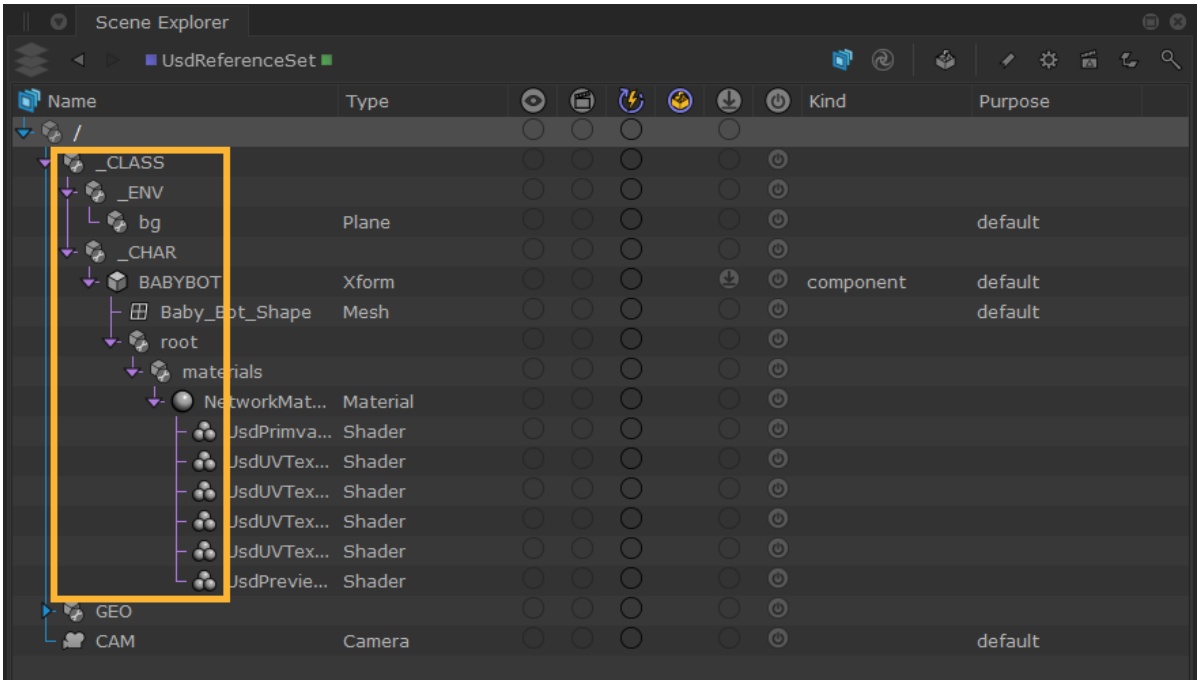
Additionally, font styling is used to indicate the following:

- **Over** prims in undefined locations show a **dimmed** font
- **Inactive** prims show a **dark strike-through** font

Abstract prim - Example:

Abstract prims can be locations under a **class**, such as the example seen below, and are often utilised to tentatively bring assets in the scene, to be later placed on the stage. A **class** prim defines a prim meant to act as a template for other prims. They ensure uniformity and reusability throughout the scene by allowing other prims to reference, inherit or specialise it, to inherit its characteristics.

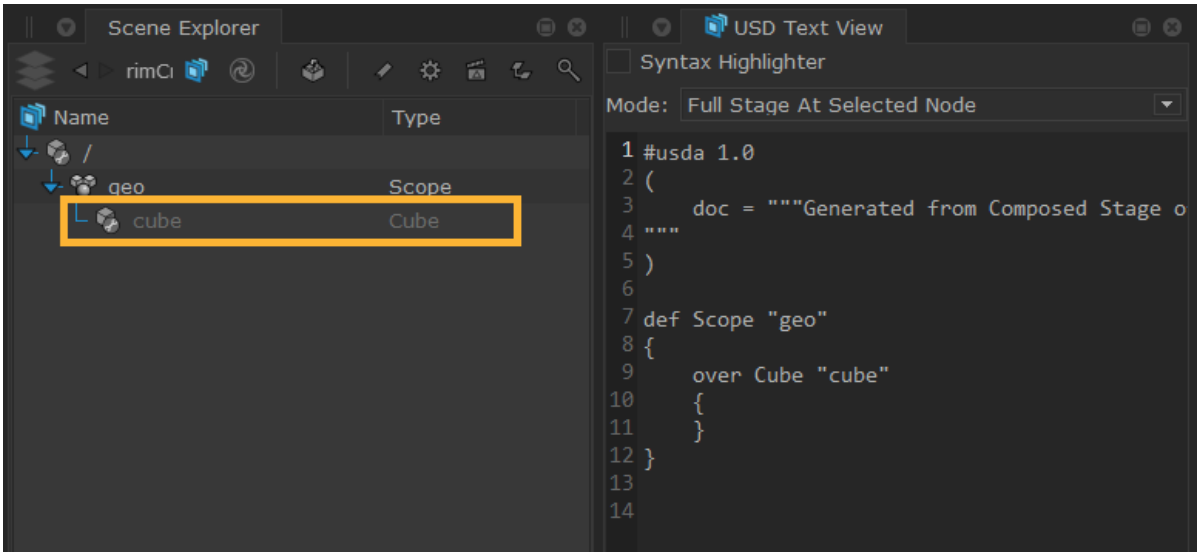
Abstract prims will be present in the usd document, however will not be present on the stage. Therefore abstract prims indicated via **purple** hierarchical lines in the **Scene Explorer**, cannot be previewed in the **Viewer** tab.



Note: This image showcases an example that can be found under **Help** > **Example Projects** > **Native USD** > **Importing and Building USD stages.**

Undefined over - Example

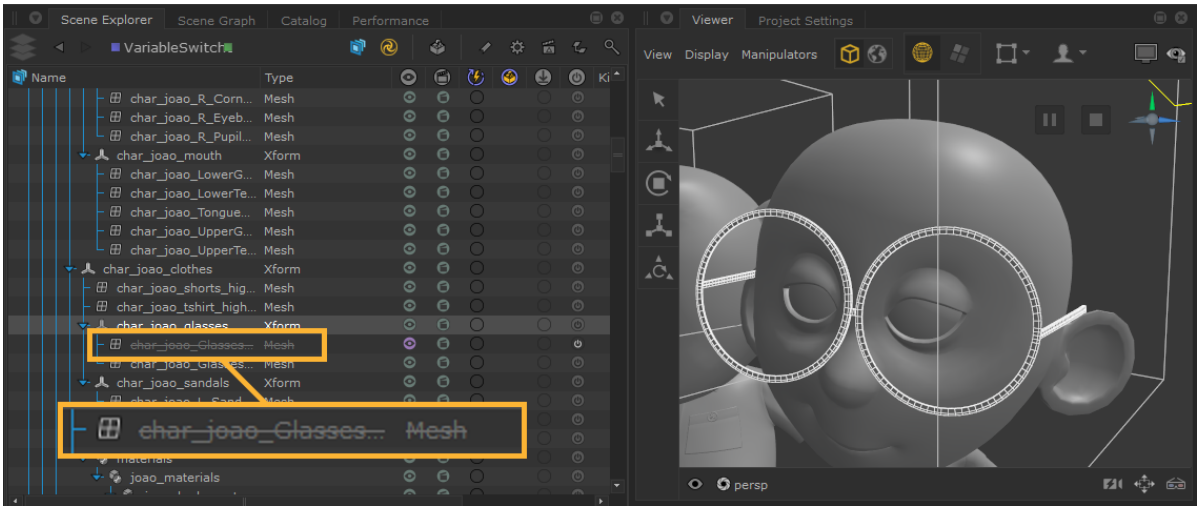
An **undefined over** is an override made on a primitive that is not defined. Here we are doing an override on a cube, but this cube was not previously created with the "def" **primSpec**. These **undefined overs** are shown in the **Scene Explorer** as **dimmed**.



Inactive locations - Example

Inactive locations will be seen in the Scene Explorer as **dimmed** and **strike-through**.

In the example, the lenses of the glasses have been set to **inactive** in the incoming USD document. This is indicated via the Scene Explorer, the same way that pruned locations are seen in Katana's geolib's panel in the Scene explorer.

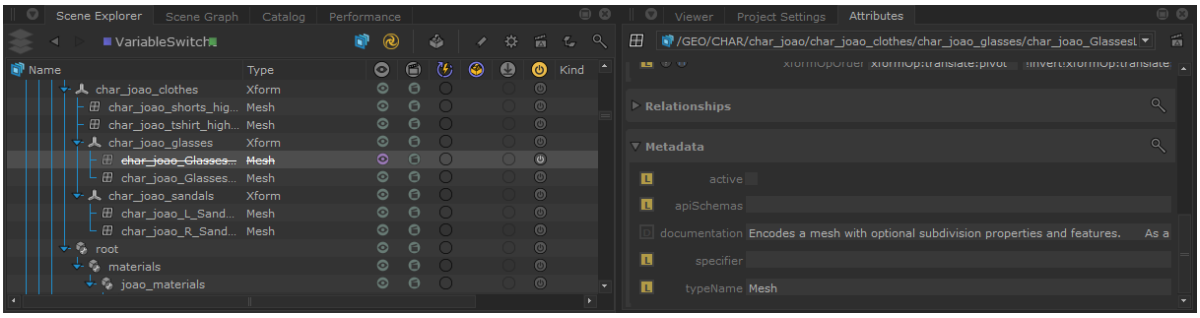


Note: Locations in USD cannot be deleted! The closest thing it has to removing elements permanently is by setting the active property on the prim location.

Another way you can identify inactive prims, is through the **Active working set** column . The icon against the **Active** column will have changed if a prim is inactive:  See Activating and Deactivating Prims.



Another way to see whether a prim is active, is through the **Attributes** tab, under the **Metadata** section. See Inspecting USD Properties Using the Attributes Tab.

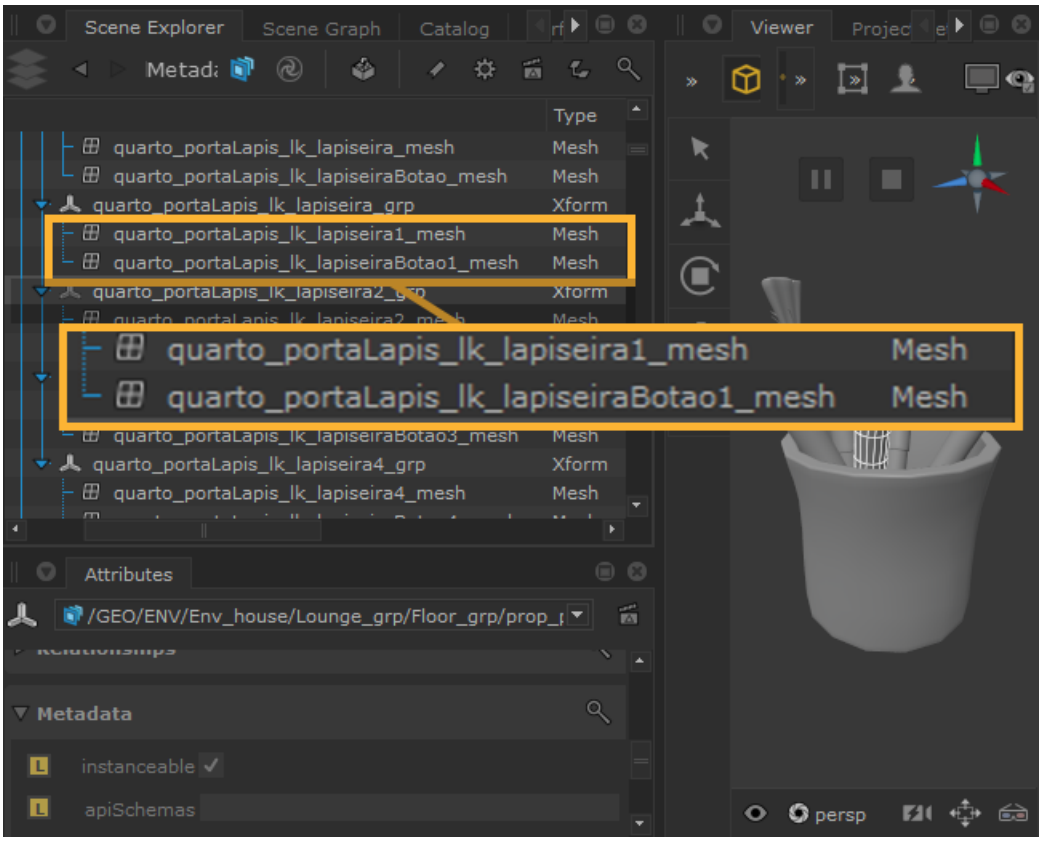


Tip: The UsdActiveSet node can be used to make a location either inactive or active, this will persist in the scene and output USD document. Alternatively, the actions made in the Active working sets within the Scene Explorer are temporary.

Instance proxies - Example:

Additionally, prims that have the property **instanceable** set to true, will be indicated in the Scene Explorer via **dotted lines**. This is to indicate prims that are an instance (an uneditable copy) of another prim location. This means any changes made to the instance proxy prims (prims that are downstream) will be ignored.

In this example, the pen and the pencil pot are all derived from the same prim, and therefore are all instances of that prim. This is a great way to build your stage in USD for prims that are repeated many times, as it can greatly improve performance.



To learn more about Scene Graph Instancing, refer to Scenegraph Instancing in OpenUSD's USD Glossary.

To learn more about Prim Specifiers, see Specifier in OpenUSD's USD Glossary.

Payloads in the Scene Explorer

Payloads for Selective Stage Loading

Katana makes use of **deferred loading** to improve performance. When working with large datasets, portions of the data can be kept in reserve until needed, thus freeing up computational resources for more pressing tasks. In the Katana Scene Graph this is achieved by selectively expanding branches of the tree. In the USD Scene Explorer, it is achieved using **payloads**.

Loading and unloading payloads triggers the recomposition of the USD stage. When a payload is loaded, it brings in new data, such as prims and properties to the stage. Similarly, when you remove a payload from a stage, recomposition is triggered as data is removed from the scene graph.

A **payload** is a reference to scene information that can be loaded into the stage when needed. Any prim can contain one or more payloads, or have none.

For example, imagine you have a tree asset that is highly detailed. You want to include multiple instances of the tree in a forest.

Your tree asset looks like this:

```
#usda 1.0
def Xform "Tree" {
    def Mesh "Trunk" { /*... mesh details ...*/ }
    def Mesh "Leaves" { /*... mesh details ...*/ }
    // ... Perhaps more variations or details like fruits, flowers, etc.
}
```

To include trees as payloads in your forest, the USD file could be written as follows:

```
#usda 1.0
    ()

def Xform "Forest"() {

    def Cube "Tree1"
        (

            prepend payload = @. / tree.usda @ < /Tree>
        ) {
            double3 xformOp: translate = (0, 0, 0) uniform token[] xformOpOrder =
["xformOp:translate"]
        }

    def Cone "Tree2"(
        prepend payload = @. / tree.usda @ < /Tree>
    ) {
        double3 xformOp: translate = (5, 0, 10) // Positioned for natural tree spacinguniform token
[] xformOpOrder = ["xformOp:translate"]
    }
}
```

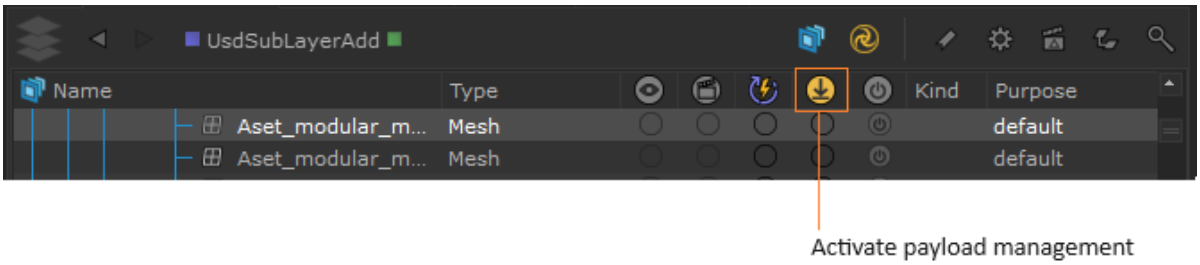
Using payload enables selective loading and unloading of specific portions of a USD stage to speed up your workflow. Loading any prims or layers that are not currently needed can be deferred until required.

Example: Consider working with a large cityscape. Instead of loading all the buildings, cars, and trees simultaneously, which can be resource-intensive and slow, you can use payloads to load only the parts of the city that are currently visible or necessary for the scene. As you shift the focus of your work, you can revise the payload loading options to select the parts you need.

Deferred loading should be familiar to experienced users who have worked with the Katana Scene Graph. As part of the USD workflow, we have incorporated deferred loading using payloads into the Scene Explorer.

Loading and Unloading USD Payloads

Payloads are activated by selecting the Payload button in the Scene Explorer toolbar.

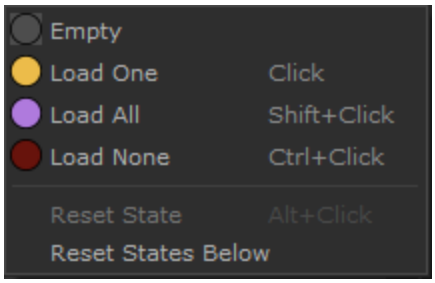


When this payload button is deactivated, payloads are loaded via expansion of USD scene graph locations, mimicking Katana's deferred loading workflow. When expanding prim locations using the deferred loading method, payloads will load if they are present on that prim, but will not load any payloads on its decendants.

Prims with payloads have a down arrow icon in the Payload column.

Payloads are managed by assigning a rule to a prim. This rule determines if payloads are to be loaded for the prim and its descendants.

There are three stage load rules available, as shown in the image below. Either right-click the dot for the options or use the mouse-click combinations listed.



Right-click in the payload column to open the context menu.

Note: Though named differently, the three stage load rules map directly to the stage load rules found in USD. Load One is the OnlyRule, Load None is the NoneRule, and Load All is the AllRule.

The rules only apply if there are payloads on the stage. When there are no payload the rules have no effect.

Prim Payload Status

The color of both the arrow and the background change to indicate the payload status on a particular prim. The table below shows the possible states.

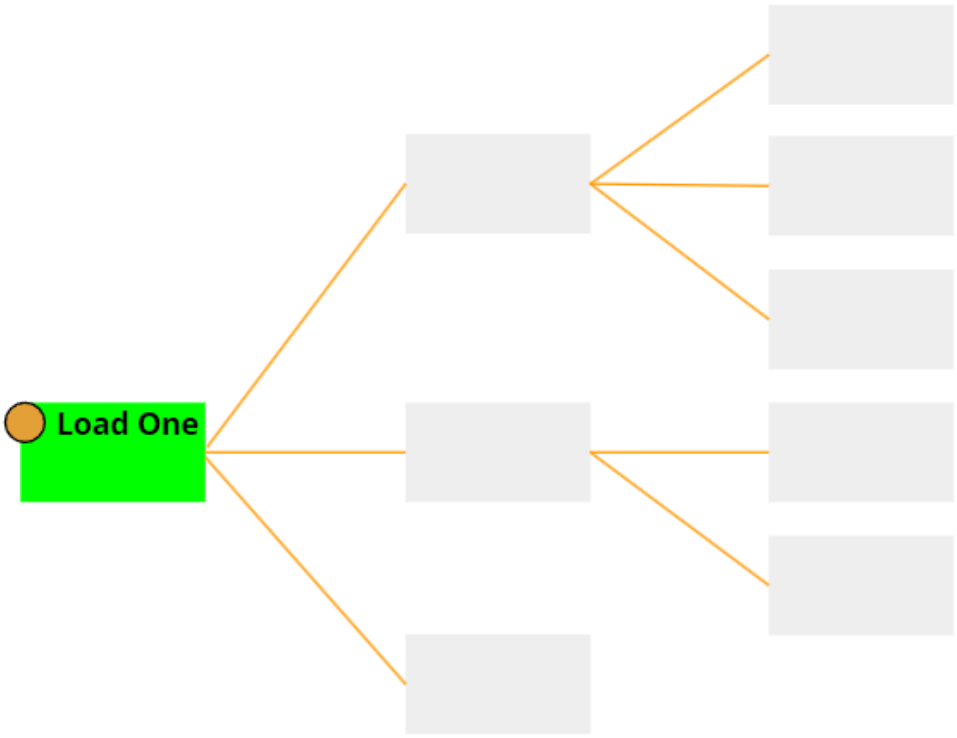
Payload Rule	Payload Management Column Active	
	Loaded	Unloaded
Empty		
Load One		
Load All		
Load None		
By Inheritance (loaded due to implicit loading by ancestors or descendant load rules)		n/a

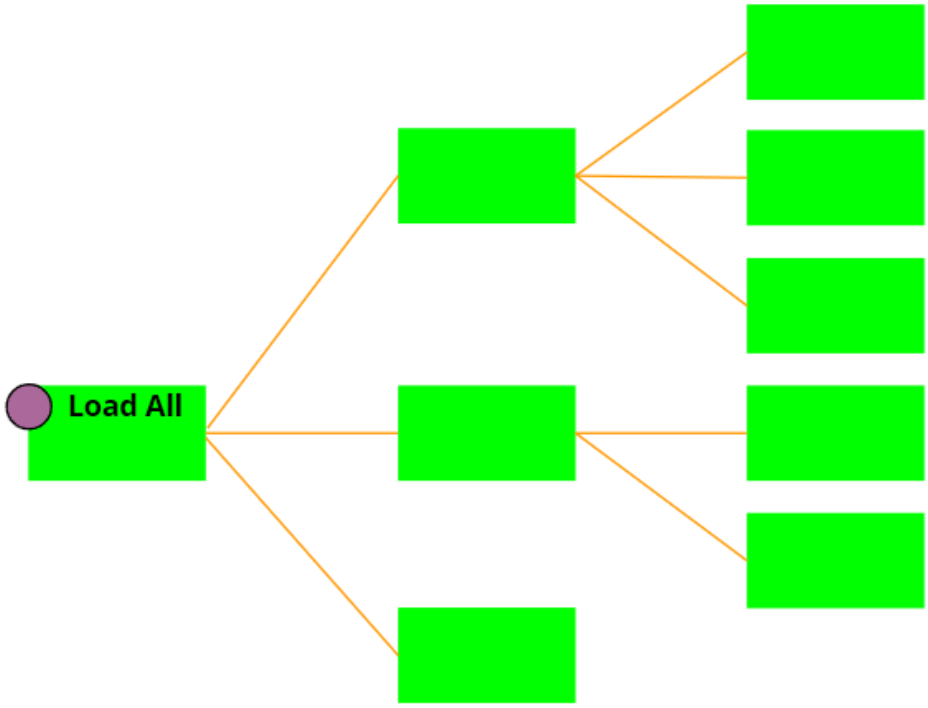
If the prim has been deactivated, Katana desaturates the icon and draws it in a grey circle.

Effects of Stage Load Rules

The table below describes how payloads are managed according to the stage load rule on the prim.

To help understand how the rules affect payload loading, the tree diagrams in the table below represent some simple scenes. Each box is a prim and the connecting lines show the prim hierarchy. A green box indicates if there is a payload at this prim, then this payload is loaded. A grey box indicates if there is a payload at this prim, then the payload is not loaded.

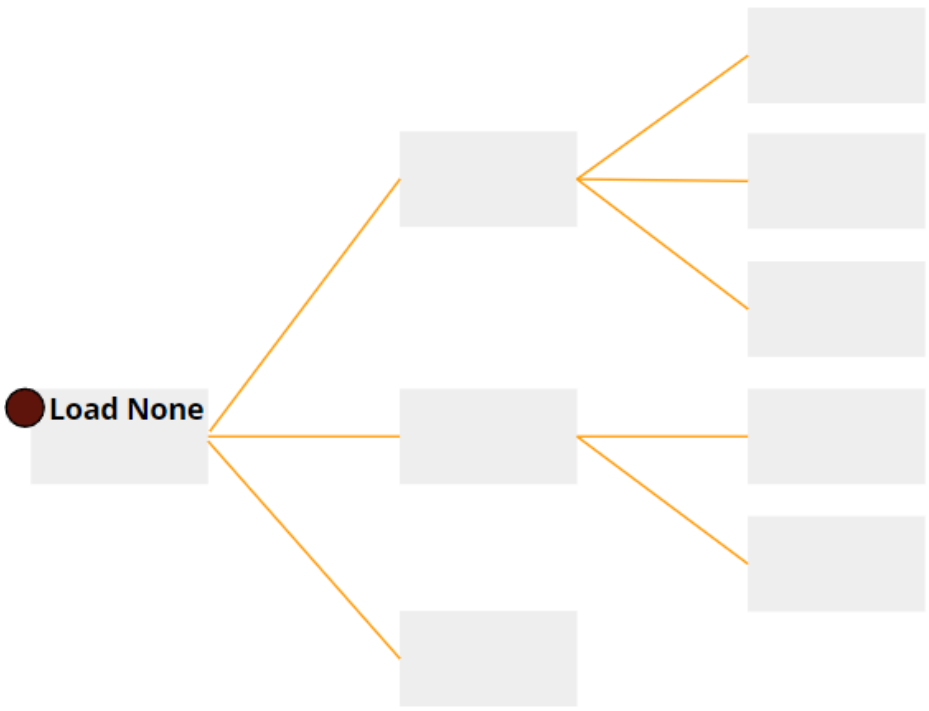
Load One Click to toggle	Load payloads on the prim but not its descendants  Only the payload on the prim in green is loaded. Example: Suppose the scene is In an old church, full of details like candlesticks, icons, stained glass windows, and furniture. You choose to work initially on the daylight shining through the windows. By using the Load One payload method you load only the structural components and windows, reducing memory overhead and enhancing viewport performance. This allows you to fine-tune the primary light dynamics without the computational overhead of secondary assets like books and artifacts.
Load All Shift + click	Include payloads on the prim plus all its descendants



Payloads on the prim and on all descendant prims are loaded if present.

Example: In the church example above, once you have addressed primary lighting behaviour, you could then use Load All to bring in all other objects and address more detailed lighting challenges.

Load None **Ctrl** + click	Exclude payloads on the prim and all its descendants.



No payloads are loaded on the prim and any descendant prims with payloads.

Example: In any building environment, such as the previous church example, there may be detailed items that you don't need to see to address a particular lighting task. Load None gives you an easy way of turning off loading for entire sections of your scene that are not in your area of focus. Aside from geometry, this could be any kind of data that you don't need, such as animation data.

Empty

the default value, or click until it resets to this value

If no rule is assigned and the prim has a payload, then the behavior depends on the load rules on the prim's ancestors and descendants.

For example, load the payload if any of the following is true:

- there are no load rules on the stage
- there are no load rules on any of the prim's ancestors
- the closest* ancestor with a rule, has the Load All rule
- the closest* descendant with a rule, has a Load All or a Load One rule

The payload is not loaded if:

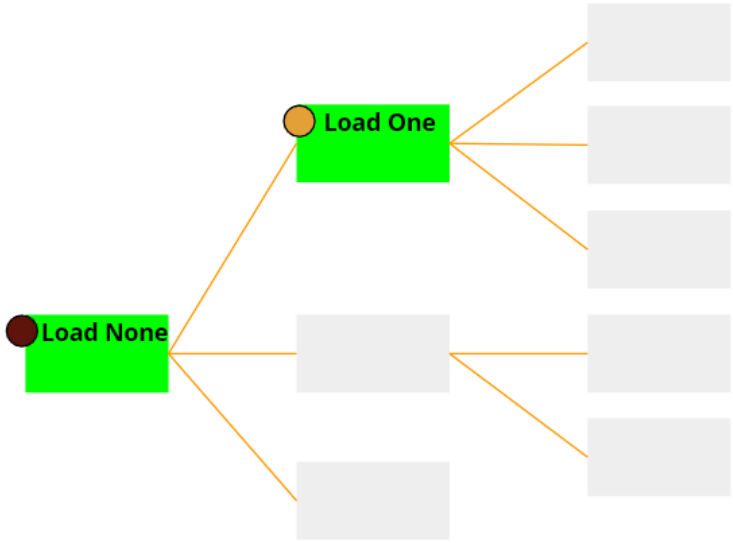
- the closest* ancestor with a rule, has a Load One or Load None rule.

(*closest can be any number of generations away, but the intermediate generations have no rule assigned)

Cascading Payload Rules to Refine Prim Loading

In real-world production scenarios, it may be preferable that certain parts of a scene should not be loaded to conserve memory. However, specific sections might still be required for a particular shot, sequence, or operation. With USD, child prims can override their ancestors' load policies, giving artists and developers precise control to optimize scenes for their specific requirements.

To achieve this, you can apply rules at different levels of the hierarchy to include or exclude specific prims as needed. For example, the tree below shows the effect of combining two rules.



A combination of payload rules on a simple graph. The leftmost node has a Load None rule but if present the payload would be loaded as there is a downstream Load One rule.

Activating and Deactivating Prims

When managing a scene, there are times when you need to toggle a prim's presence to remove it from the scene.

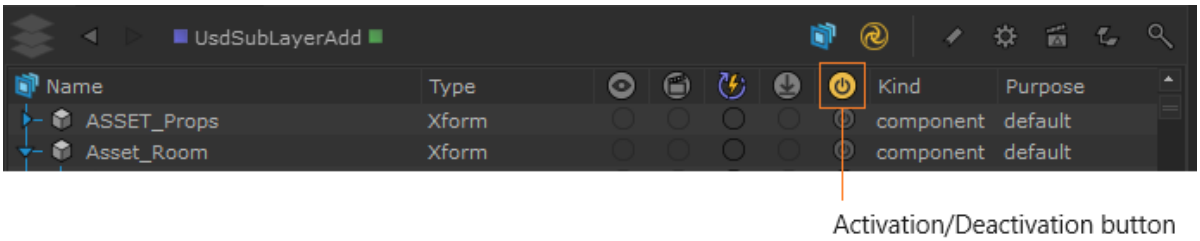
USD does not offer a way to delete primitives, as it is built for non-destructive editing. However, you can simply deactivate prims so they are hidden from the stage and have no effect on the result, removing them from stage composition.

The working set option in the Scene Explorer is a temporary action that will allow to toggle on/off per prim level. However, be aware that if these changes are not reflected in the usda file (i.e. incoming assets or toggled within Katana's node), then it won't actually be exported as part of the document.

Note: Since the "active/inactive" state is a Property in USD, you can also modify the active state using a UsdActiveSet node, which will add this property in Katana to any prim you'd like to remove.

Example: You may often have multiple layers contributing to the final scene. Activation/deactivation enables non-destructive overrides in more influential layers. For instance, a prim present in a base layer can be deactivated in an override layer, changing the final scene without altering the original content.

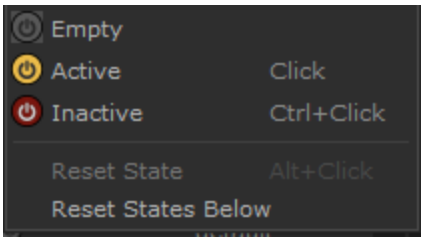
You can enable selective pruning by activation and deactivation using the Activation button in the Scene Explorer toolbar.



Click the power style button to enable activation/deactivation.

With the Activation column toggled on, you can easily deactivate and re-activate prims temporarily using the button in the column.

Click on the button or right-click to open the context menu.



With stage activation on, prim activation can have one of four states:

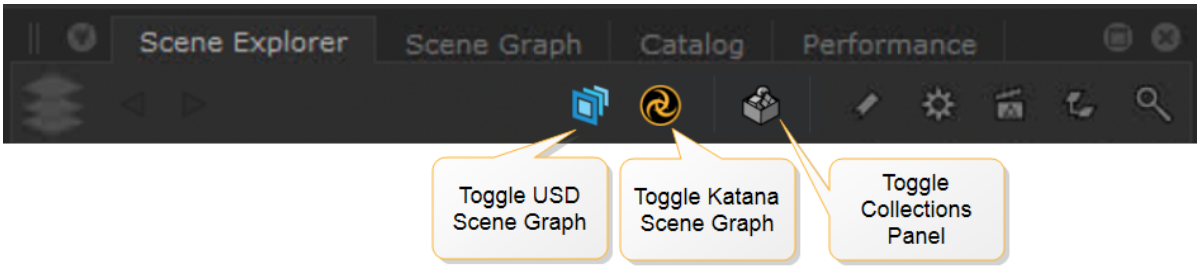
Status	Icon	Info
Empty (Default)		No active state has been assigned to the prim by the Scene Explorer, and the prim is active in the stage.
Inactive Set by USD or node		No active state has been assigned to the prim by the Scene Explorer, and the prim is inactive in the stage.
Active Click to toggle		The prim is set to active by the Scene Explorer. All the descendants are active unless otherwise deactivated.
Inactive Ctrl + click		The prim has been deactivated by the Scene Explorer, its descendants are not included in the stage.

Note: There may be other combinations due to the interplay of the Scene Explorer Active column control with the USD, but inactivity is essentially indicated by a light-grey power symbol. A color-filled circle means that the active status is being controlled by the Scene Explorer.

Collections in the Scene Explorer

There is now a panel above the scene graphs, showing the **Collections** in the project. This is collapsed by default.

Use the three selection buttons at the top of the Scene Explorer tab to display or hide the USD Scene Graph, or/and the Katana Scene Graph, or/and the Collections Panel.

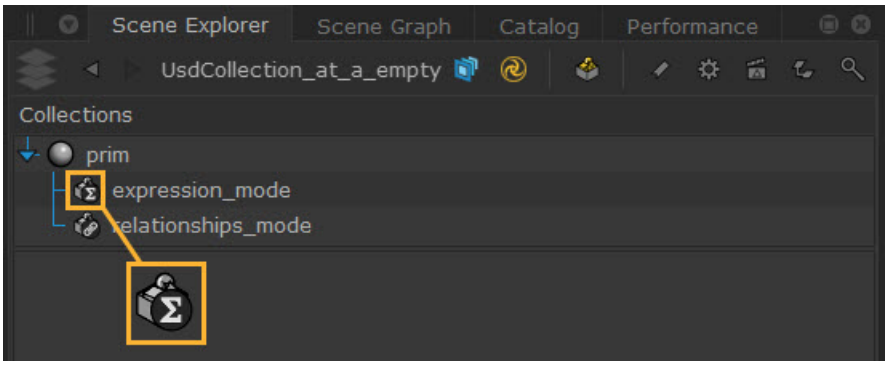


If a panel is hidden, its icon is greyed-out.

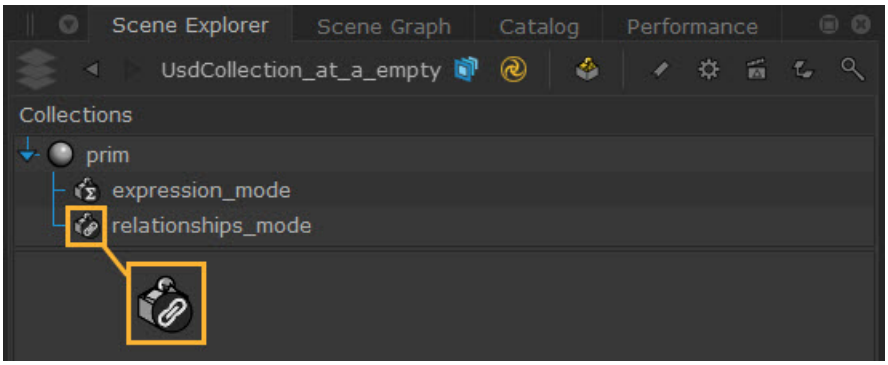
The **Collections Panel** is split into Katana and USD sections. Hiding the Katana or USD section of the main Scene Explorer tab by clicking the USD Scene Graph and Katana Scene Graph toggle buttons also causes the respective section of the Collections panel to hide.

Collections Panel

Expression mode collections set by a membership expression appear in the Collections Panel with the icon:

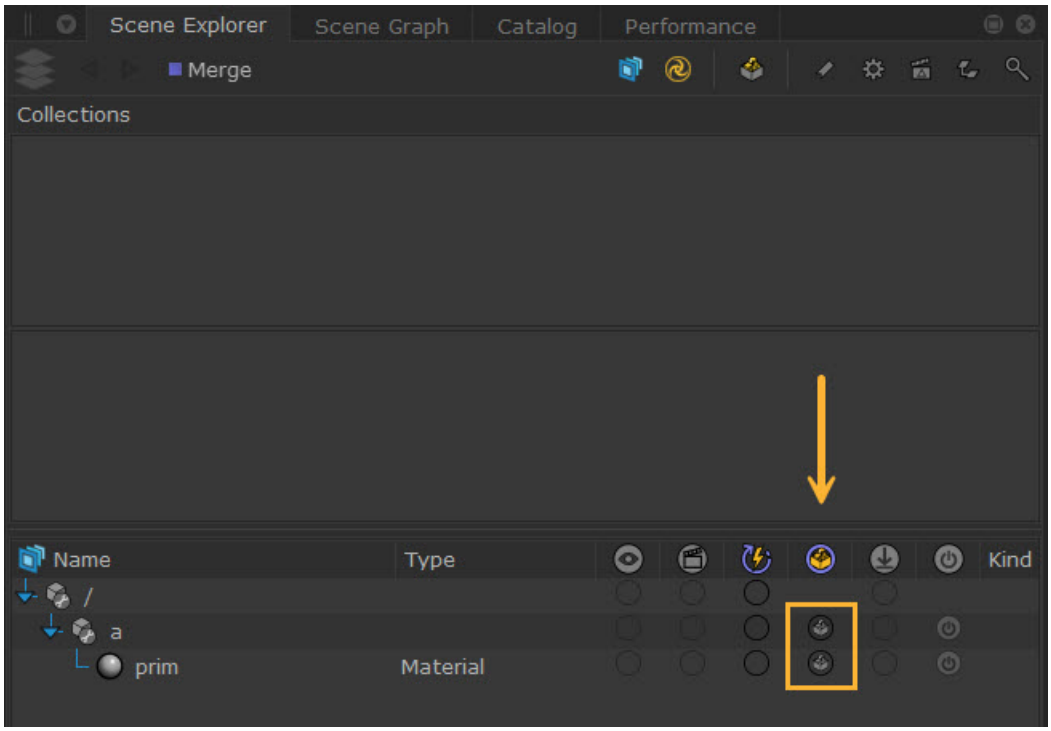


Relationship mode collections set by specifying includes/excludes appear in the Collections Panel with the icon:

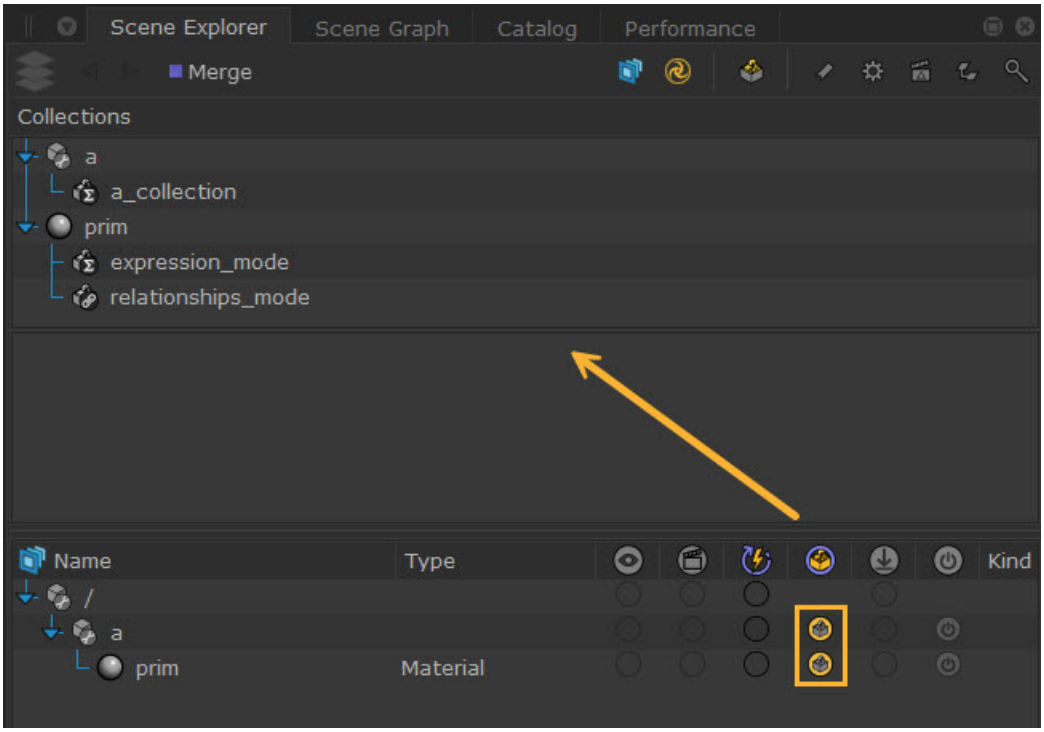


Using the Collections Working Set

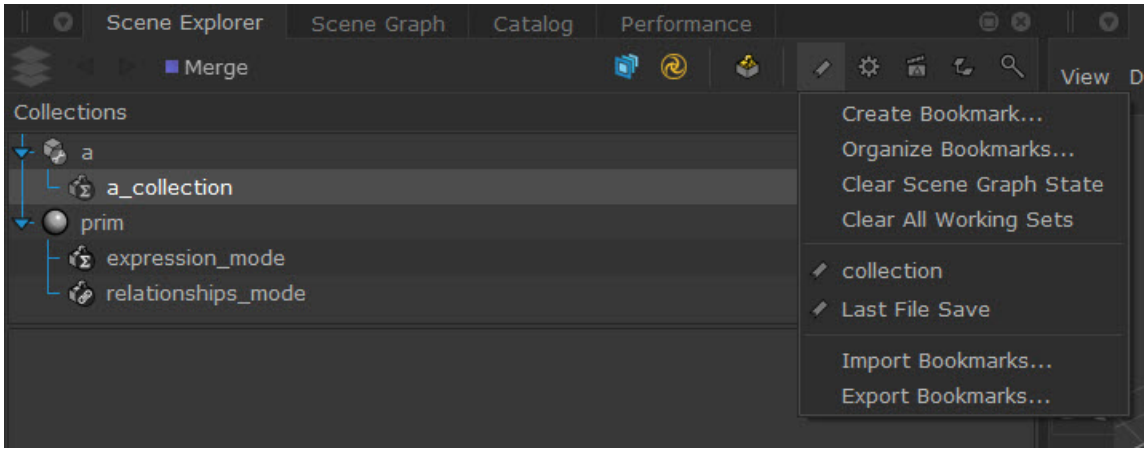
The Collections working set shows which locations have a collection on them.



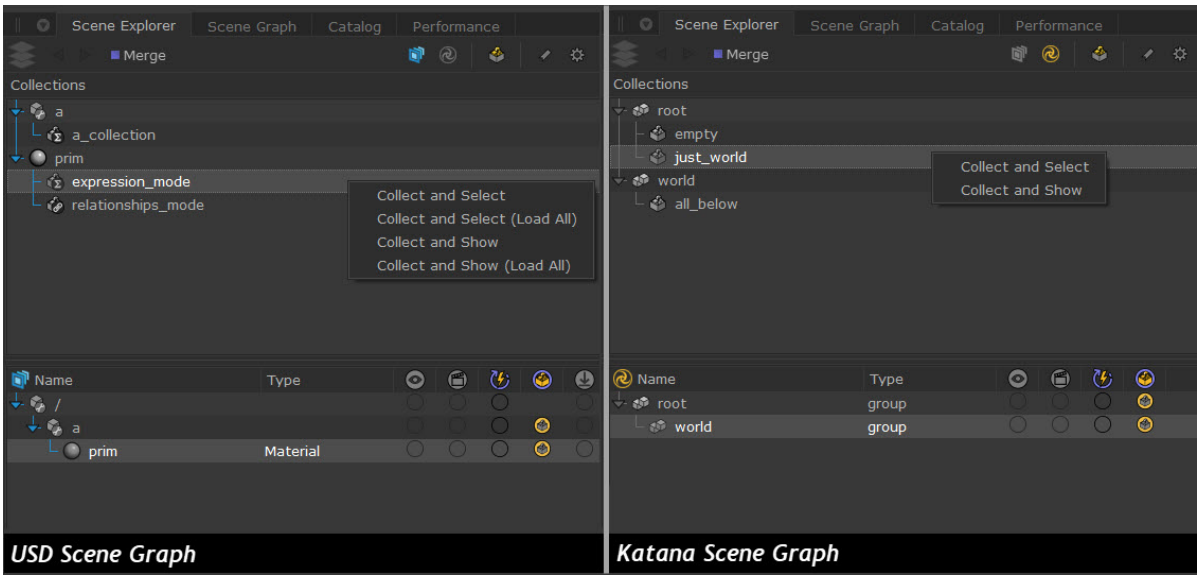
- To view which collections are available, click the **Collection icon**  next to a location in the Collections working set column. Once selected, the collection is displayed in the Collections panel above.



• Any selected collection in the working set can also be saved as a bookmark.



Selecting and Showing Collections



- Right click on a collection in the Collections panel and choose **Collect and Select** - this expands the branches and automatically selects them in the Scene Explorer.
- Right click on a collection in the Collections panel and choose **Collect and Show** - this displays all the paths contributing to that collection in a new window.

Only applicable for USD Collections

- Right click on a USD collection and choose **Collect and Select (Load All)** - this loads its Payloads while expanding the branches and automatically selects them in the Scene Explorer.
- Right click on a collection and choose **Collect and Show (Load All)** - this loads its Payloads while displays all the paths contributing to that collection in a new window.

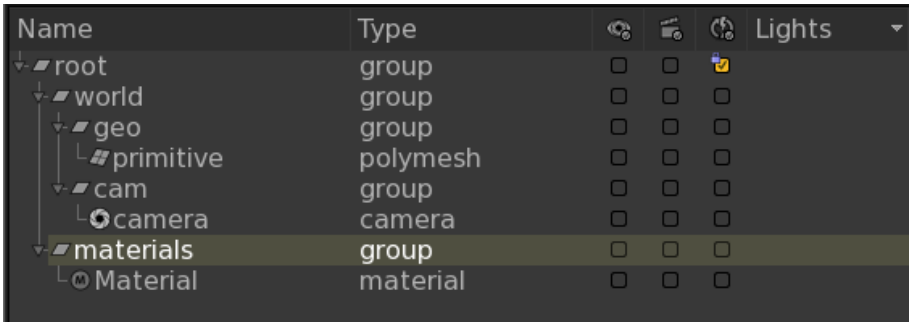
For more about Collections, see Pattern-Based Collections with USD.

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



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

- Parent - the location **/root** is the parent of **/root/materials**.
- Child - the location **/root/materials/Material** is a child of **/root/materials**.
- Sibling - the location **/root/world** is a sibling of **/root/materials**.
- Leaf - the location **/root/world/geo/primitive** is a leaf location. A leaf is a location with no children.
- Branch - 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

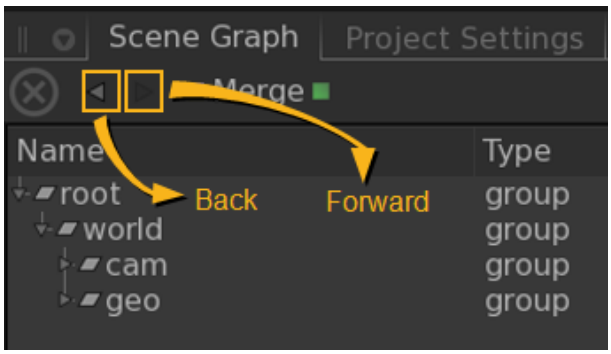
Click within the faint square to the left of the node.



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.



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.

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  .

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 3Delight or Renderman. 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 render 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 one to create new filters is the C++ Scene Graph Generator API.

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

To select 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.

To select the parent of the selected location(s):

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

To select the children of the selected location(s):

1. Right-click on the selected location(s).

2. Select **Select** > **Select Children**.

To select the leaves of the selected location(s):

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

To invert the selection with its siblings:

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

To select 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.

To deselect a location, **Ctrl**+click on the location.

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.
   - Search for nodes by their **Name** or **Type** by switching between the two options in the dropdown next to **Filter**.
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.

The **Type** filter also filters nodes by their **nodeType**. When dealing with renderer-specific nodes, such as Shading Nodes, the node's **Type** may not be the same as its **NodeType**. For example, the **Type** for a DlPrincipled Node is DlShadingNode, while its **nodeType** is dlPrincipled. The **Type** parameter defines the node as being a 3Delight node, while the **nodeType** defines what the node's function is within a shading network. Seeing a node's **nodeType** allows you to quickly see which Shading Nodes are present in your shading network and how they may be contributing. This applies to any shading nodes belonging to third party render vendors such as Arnold, USD or RenderMan.

Expanding the Scene Graph

To expand 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.

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.

To expand the Scene Graph to a limited level:

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. 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.

- **Expand To** > **Viewer Visibility**

  Expands the scene graph until it reaches a Viewer Visibility working set location with a non-empty state.

- **Expand To** > **Render**

  Expands the scene graph until it reaches a Render working working set location with a non-empty state.

- **Expand To** > **Live Render Updates**

  Expands the scene graph until it reaches a Live Render Updates working set location with a non-empty state.



Note: You can also right-click on a location and select **Expand To and Select Proxy Children** to reveal scene graph locations that provide proxy data.

To expand the Scene Graph location to 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

To collapse a location and all its children:

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

To collapse a Scene Graph location:

• Click  next to the location name.

OR

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

To collapse 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.

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.

  This proxy Op chain is always evaluated in isolation, starting at the **/root** location of an empty scene graph.

  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.

You can also right-click on a location and select **Expand To and Select Proxy Children** to reveal scene graph locations that provide proxy data.

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**.

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 that 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 a 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 an Op Type 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.

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. through the **ScenegraphManager Python** module, which maintained a global expansion and selection state for the whole application. 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 also provide a reusable API for similar cases that need a definable set of target locations in the UI.

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.

Technically speaking, a Working Set is a set of scene graph locations for which membership is defined by a set of location states. There are four explicit states: **Included**, **Included with Children**, **Excluded**, and

Excluded with Children. A scene graph location is a member of a Working Set if it is explicitly included or if it inherits inclusion from an ancestor that is **Included with Children** and is not explicitly excluded and does not inherit exclusion from an ancestor that is **Excluded with Children**. Working Sets are independent of the existence of scene graph locations, and their expansion states.

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.

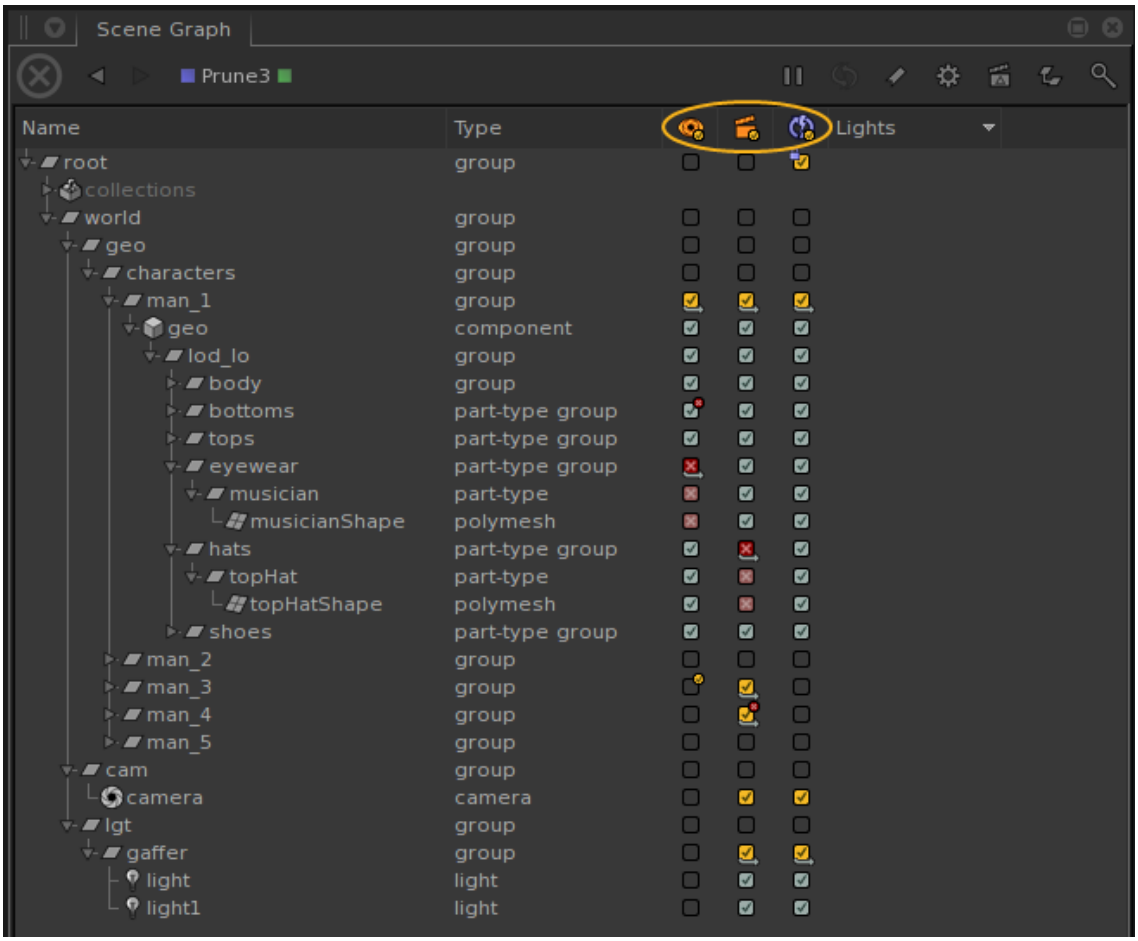


Note: You can disable the Working Sets UI elements, including the **Viewer Visibility** and **Render** columns in the **Scene Graph** tab and the corresponding buttons in the **Viewer** and **Monitor** tabs, by setting the **KATANA_DISABLE_WORKING_SETS_UI** environment variable to `1`.

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 that 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.

The **Viewer** tab works in two modes:

- **Scene graph expansion** - visibility depends on scene graph expansion state and pinned locations.
- **Visibility Working Set** - locations included in the Visibility Working Set.



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: If the **Render Only Selected Objects** toggle is turned on and the Render Working Set is enabled, only locations that are both selected and contained in the Render Working Set are rendered. For more information, refer to Rendering only Selected Locations.

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.

Revealing Locations with Working Set States

You can expand the Scene Graph, through the right-click context menu, to a working set location with a non-empty state:

1. In the Scene Graph, right-click on the location to expand.
2. Select one of the following working sets to expose:
   - **Expand To** > **Viewer Visibility**

     This expands the scene graph until it reaches a Viewer Visibility working set location with a non-empty state.
   - **Expand To** > **Render**

     This expands the scene graph until it reaches a Render working set location with a non-empty state.
   - **Expand To** > **Live Render Updates**

     This expands the scene graph until it reaches a Live Render Updates working set location with a non-empty state.

Including Proxy Children in Working Sets

In the Scene Graph, you can include locations with proxies attributes into the respective Working Set. To do so, in the **Viewer Visibility**, **Render**, or **Live Render Updates** column, right-click on one or more selected locations and select **Include Proxy Children...**

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.

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 .

To pin 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).

To pin all the visible leaves, first ensure the locations you wish to pin are expanded in the **Scene Graph** tab, then:

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.

To clear 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).

To clear pins below a 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).

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**.

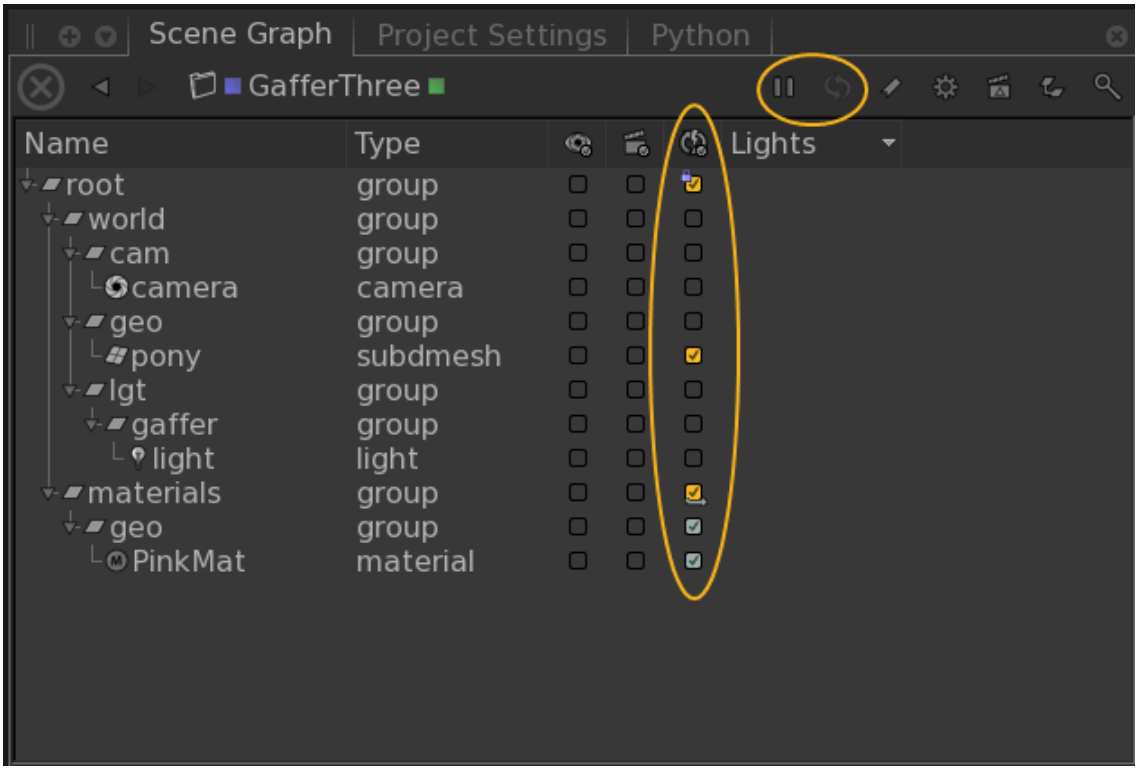
   A file containing all the exported bookmarks is saved.

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.

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.

3D node parameter values are finalized with all pending changes prior to performing a render. For more information on the **3D Update Mode**, see Global Options in Rendering Your Scene.

Making Use of Different Location Types and Proxies

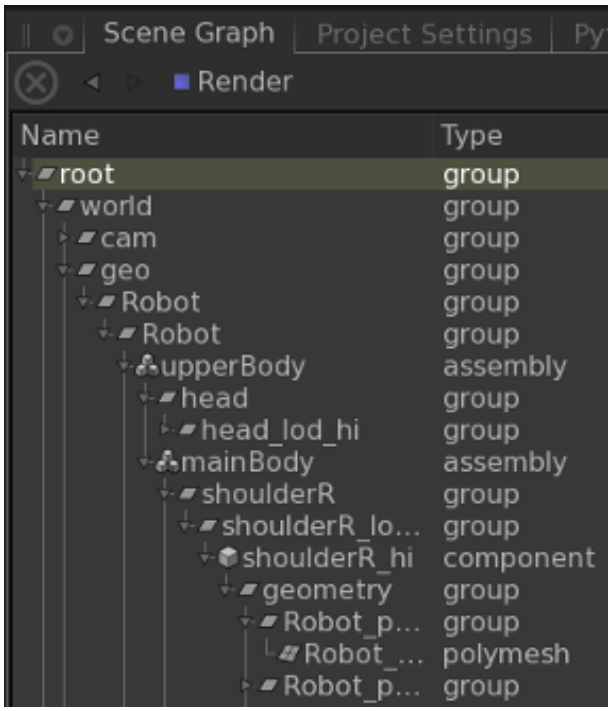
Only loading what is needed, when it is needed, is a key part of the philosophy of Katana. If you want to position the specular highlight on your main character, for instance, you don't need to load the entire scene. One way to avoid unnecessary loading is to define your scene with special hierarchies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

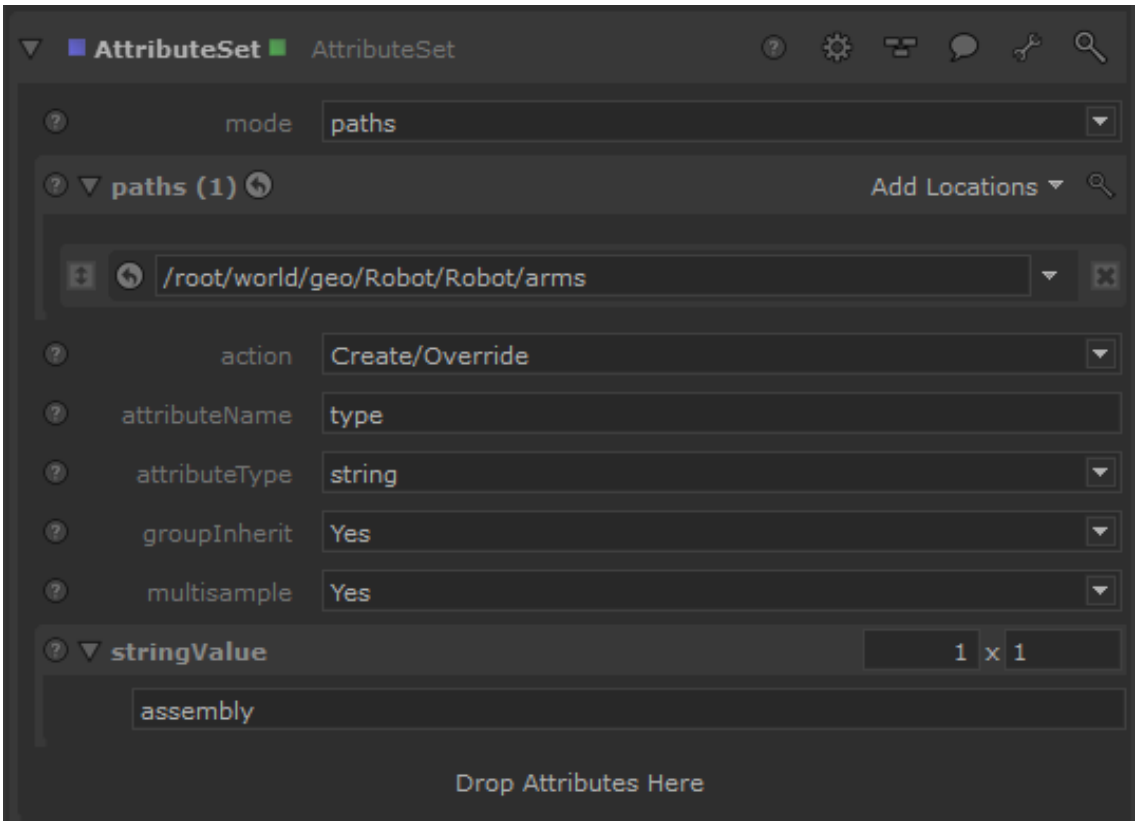
Proxies enable you to get a good idea of a scene without opening up too much of the hierarchy. Placing proxies on assemblies and components enables you to open a hierarchy to convenient levels of scene complexity.

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**.

Resolvers

Resolvers are Ops 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 the can be passed from one Op 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 OpScript to set the value of **materialAssign** using a Lua script.
- Using a custom Op to set the value of **materialAssign**.

You can also use a LookFile that resolves the correct value for **materialAssign** onto given objects.

The LookFile system has special behavior for handling materials that have been assigned to an asset. This allows other users to make edits and overrides to material values, while also keeping processing efficient for the majority of cases where there isn't any need for modifications. This behavior is also designed to handle cases where there may be more than one version of a look file used in a shot and where there are multiple output passes, and overrides may need to be applied to any selection of passes. Renderer procedurals are also handled in a similar manner to materials.

LookFileResolve is designed to look for the existence of materials (and renderer procedurals) at specific locations in the scene graph. If materials are found at these locations then **materialAssign** attributes are set at locations in the asset that use those materials. This allows you to make modifications, such as material overrides, to all objects that use a material from the LookFile. If no material is found at any of the search locations then the materials from the LookFile are directly applied to locations in the asset without using **materialAssign**.

The main reason for this system is efficiency: most materials on most assets probably don't need any modifications, such as overrides, and it is more efficient to simply paste the relevant attribute values onto the asset. However, if you want to apply overrides then it's more convenient to bring the materials directly into the scene and set them up as assigned materials.

To avoid clashes between materials with the same name that are brought in from different LookFiles, a **namespace path** is used as the location under which the materials are brought in. This is constructed using the following options:

- A custom path (specified by a parameter on LookFileOverrideEnable) or a path provided by the asset management system based on the LookFile asset using **getUniqueScenegraphLocationFromAssetId()**.

- If the asset management system provides version numbers for look files, the path can either explicitly include the version number, which means that any overrides only apply to assets that use that explicit version LookFile, or a version-less path, which means that any overrides apply to assets that use any version of the LookFile.

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.
- OpResolve. This resolves any deferred Ops that have been set to run during an op resolve.
- LightLinkResolve. This resolves the attributes, which the LightLinkSetup node sets on **/root/world.lightList**.

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 list of implicit resolvers is as follows:

- Preprocess Resolvers
    - **OpResolve(resolveIds=implicit_preprocess, lookfileresolve)** - this resolves deferred Ops set to resolve **during katana look file resolve**. Ordinarily, such ops are resolved by LookFileResolve nodes; this resolver is a fail safe for when a LookFileResolve node is not present.
- Standard Resolvers
    - **LightLink.Resolve** - this resolves light linking information that was previously set up by a LightLinkSetup node.
    - **ReferentialInheritanceResolve** - this resolves all locations with an inherits attribute referencing another location. The attributes from the referenced location are overlayed.
    - **MaterialResolve** - this looks for **materialAssign** attributes and creates local copies of materials taking into account any material overrides.
    - **OpResolve(material attr)** - this resolves deferred Ops set to resolve **during material resolve**.
    - **MaterialUnderlayAttributesResolve** - this resolves material underlay information by copying attributes from a location's material.underlayAttrs group to that location's root level without clobbering existing attributes.
    - **RendererProceduralResolve** - this is similar to MaterialResolve but for renderer procedurals. It looks for any locations with **rendererProceduralAssign** attributes.
    - **FilenameResolve** - this resolves `{attr:xxx}` tokens in **material** and **rendererProcedural** attribute groups. Refer to Using the {attr:xxx} and {globalattr:xxx} Syntax for Shader Parameters section for more information.
    - **BoundsAdjust** - this resolves any deferred bounds padding task that was previously set up by a BoundsAdjust node.
    - **BoundsPropagateToAncestors** - this resolves any deferred bounds propagation task that was previously set up by a BoundsAdjust node.
    - **AdjustScreenWindowResolve** - this resolves any screen window adjustment previously set up by a RenderSettings node.
    - **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.
    - **MuteResolve** - resolves mute states for lights created by GafferThree nodes.
- Postprocess Resolvers
    - **OpResolve(resolveIds=all)** - this resolves deferred Ops set to resolve **during op resolve**.
    - **RenderSettingsLocalize** - this resolves render resolution, sample rate, and region of interest.

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 Scene Graph 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.



Note: See more in the Katana Developer Guide: Attribute Conventions - Ops and Proxies.

Creating Your Own Resolvers

Custom Resolvers in the Node Graph

You can use OpScripts or custom Ops to create your own resolvers, including having them run implicitly.

There are a number of modes available for OpScript, GenericOp, and Op execution. These are controlled by the **resolveIds** values in the attributes. There are two modes available for the execution mode:

- **immediate** - the script/Op is run at the locations specified in the **applyWhere** parameter as it is evaluated at this node's point in the node graph.
- **deferred** - the script/Op is set up by this node but won't actually be run until a later node in the node graph, as specified by the **applyWhen** parameter.

And depending on what you choose, you have the option to set where or when the script/Op is run. When the execution mode is **immediate**, the applyWhere parameter can be set:

- **at all locations** - at all the locations in the node graph.
- **at specific location** - 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.

When the execution mode is **deferred**, the applyWhen parameter can be set:

- **during op resolve** - the script/Op and its arguments are added as attributes to be executed later by an OpResolve node. If the Op isn't run by an explicit OpResolve node placed in the node graph, it is automatically run at render time by the **implicit resolvers**.
- **during material resolve** - the script/Op and its arguments are added as attributes under the **material.ops** group attribute. This is primarily intended for material scene graph locations, allowing the material to specify a procedural process that is run at every location that the material is assigned to. The script/Op is run as part of the material resolve process, and is executed just after the initial values for the material shader are created at the location. Examples of its use include randomizing or procedural control over shader parameters.
- **during katana look file resolve** - the script/Op and its arguments are added as attributes under the **ops** group attribute and are evaluated by a LookFileResolve node or by the first implicit resolver if no LookFileResolve node is present.

For more information on OpScripts, see the Working with Attributes section. For more information on GenericOp and Op API, refer to The Op API.

Custom Implicit Resolvers

Custom implicit resolvers can also be made persistent across the Katana session, without the need for additional nodes in the node graph. The **Nodes3DAPI** module provides the **RegisterImplicitResolver()** function for this purpose, with the following signature:

```
RegisterImplicitResolver(stage, opType, opArgs, addSystemArgs=False)
```

The **stage** parameter determines where the new resolver should be inserted in the chain of implicit resolvers. The stages are described in Implicit Resolvers. Its value should be one of:

- `ImplicitResolverStage.BeforePreprocessResolvers`
- `ImplicitResolverStage.BeforeStandardResolvers`
- `ImplicitResolverStage.AfterStandardResolvers`
- `ImplicitResolverStage.AfterPostprocessResolvers`

Op types, args, and system args are described in The Op API.

Note: See more in the Katana Developer Guide: Attribute Conventions - Ops and Proxies.

Building Your Scene

This section walks you through setting up your scene in Katana.

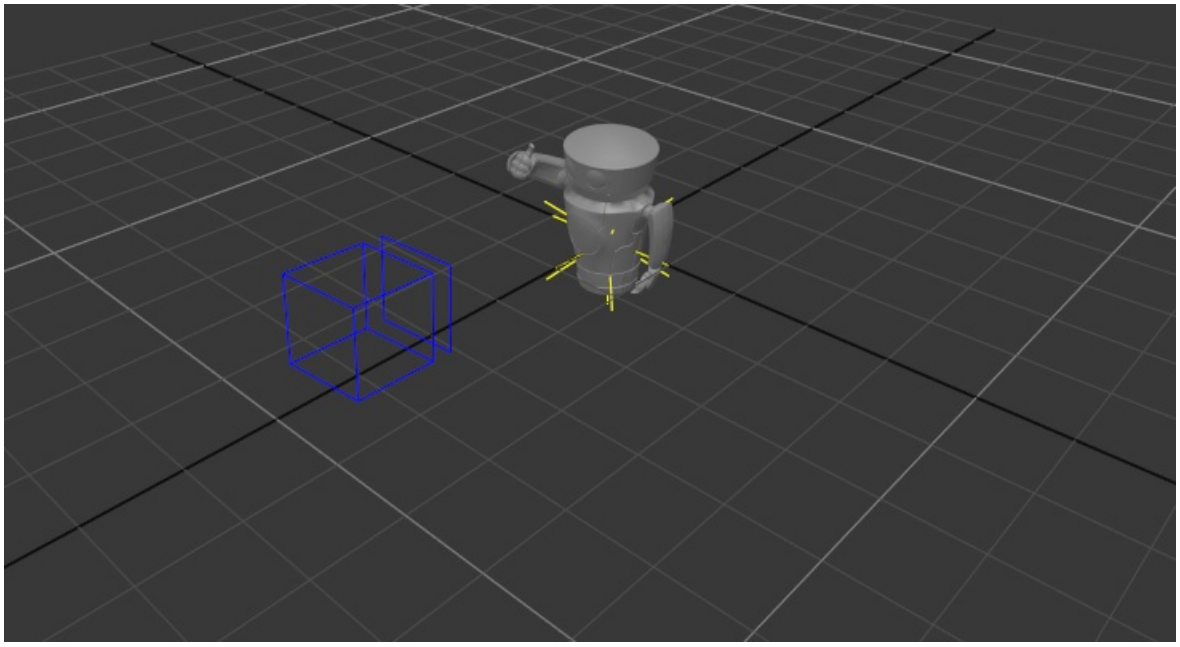
Adding 3D Assets - Usually, a recipe is started by defining the 3D assets.

Collections and CEL - Use Collection Expression Language (CEL) to describe locations in the scene graph that receive operations or assignments.

Working with Attributes - How to manipulate scene graph attributes.

Katana does not assume a given scene scale. There are no physics solvers, like those in Maya, that need to know what real world measurement a unit represents. If you're using a shader library that has real world units for shade parameters, such as emissive lights with power per unit area, that's handled by the shader implementation rather than Katana.

The unit of measure used is up to you, because Katana equates any value as being equal to one unit in 3D space. The image shows a simple scene containing an object and a camera. If the large white square in the Viewer represents one unit of measure, so the smaller squares represent one tenth of that unit. So, if you measured on set in meters, one small square could be equal to a meter, centimeter, or millimeter.



Adding 3D Assets

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.

- **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 Casting Sheets. To learn more on its use, see Using the Importomatic.

- **USD**

Universal Scene Descriptor (USD), is an open-source scene information interchange framework developed by Pixar.

- The **UsdIn** node allows you to import USD assets into Katana. To learn about about it, see Loading in USD Assets with UsdIn in the user guide.
- There are also a set of USD **native nodes** that allow you to create USD prims in Katana, see Native USD Prims.



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.

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 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 Casting Sheet** and select the asset from the file browser or asset management browser.

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



Note: See Importing Assets Using Casting Sheets for more information.

Importing Assets Using Casting Sheets

The **Add Casting Sheet** option of the Importomatic node opens a **.cast** file (casting sheet) that contains assets and names for those assets. This allows you to load multiple alembic files at once.

Below is an example of a casting sheet file:

```
<castingsheet>
    <entry assetid="/tmp/Gnome.abc" name="gnomeA"/>
    <entry assetid="/tmp/Pony.xml" name="ponyA"/>
</castingsheet>
```

The **assetid** XML attribute is an asset identifier you can use with whichever asset management system you are working with. If you are working with files then the **assetid** XML attribute is a file path. For more on Asset IDs, see Concepts.



Note: Make sure to specify a file extension in your file path for the casting sheet to determine which asset type to load.

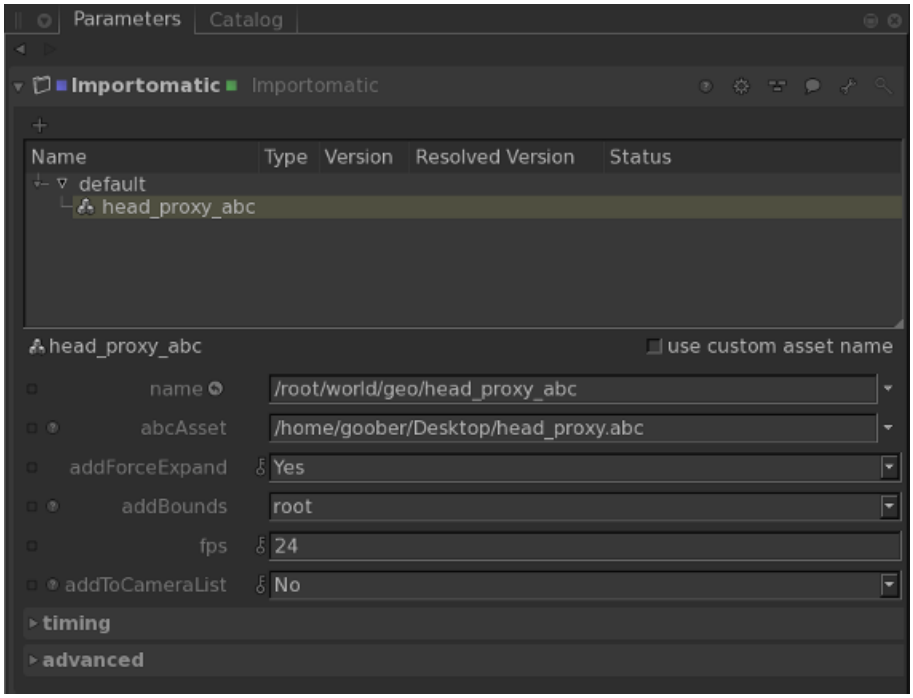
The **name** XML attribute is the identifier that is used to reference an instance of that asset in your Katana scene. You can use it to determine the default scene graph location for the asset and to label the node used to import it.

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.



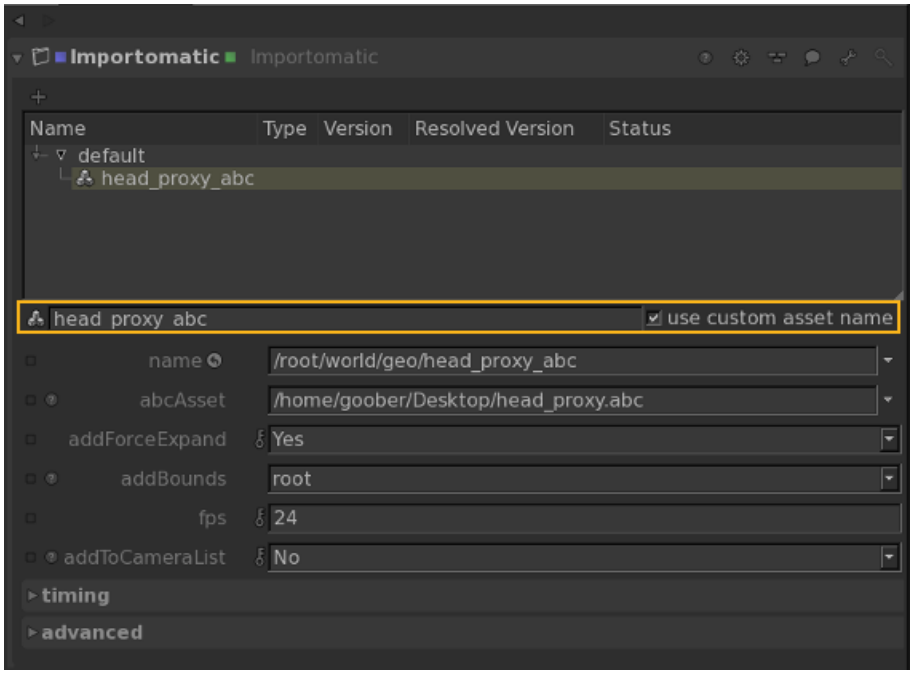
Editing an Asset's Name

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 and Enabling 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.

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:

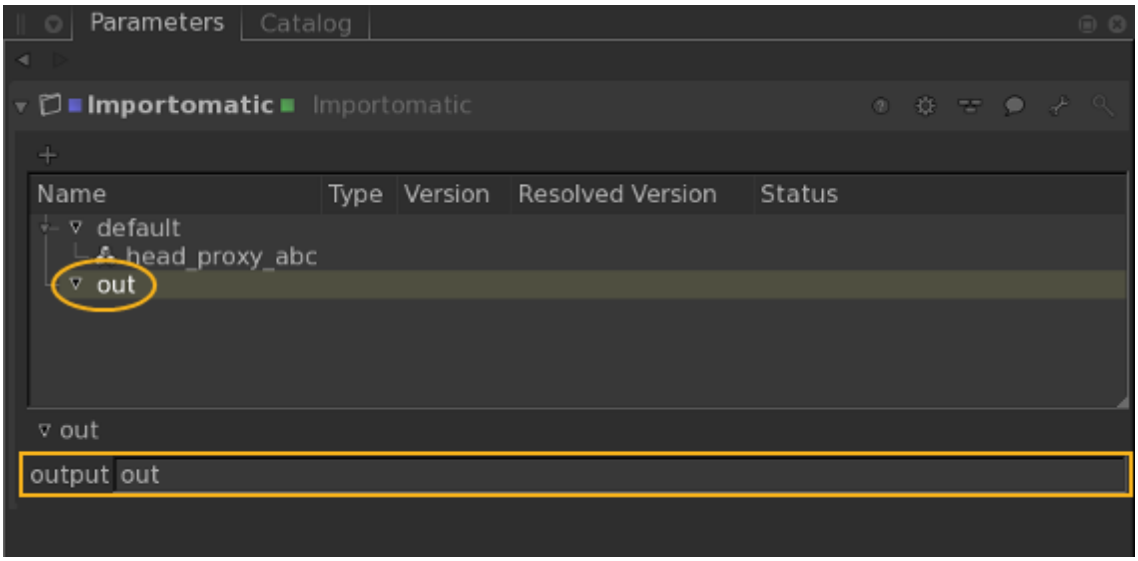
1. In the Importomatic parameters, click  .
   The asset and output menu is displayed.
2. Select **Add New Output**.
   A new output is added to the node and hierarchy.

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.



Collections and CEL

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** that 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.

See more at CEL in the User Interface.

Note: Similar to Katana's CEL, USD's Pattern-Based Collections (PBC) was introduced in USD 23.11 and is now fully integrated in Katana's USD framework, with the addition of a Pattern-Based Collection (PBC) widget and the UsdCollection node, available as of Katana 7.5. See Pattern-Based Collections with USD.

Note: Find out more about Collections and CEL in the Katana Developer Guide: CEL Reference - Collections.

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.

Note: Similar to Katana's CEL, USD's Pattern-Based Collections (PBC) was introduced in USD 23.11 and is now fully integrated in Katana's USD framework, with the addition of a Pattern-Based Collection (PBC) widget and the UsdCollection node, available as of Katana 7.5. See Pattern-Based Collections with USD.

You can see a list of the CEL and USD (PBC) Collections in your project in the Collections panel  at the top of the Scene Explorer and via the Collection working set indicators  . See Using the Scene Explorer.

Guidelines for using CEL

Use 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 inadvisable 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 that 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.

A 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 versus 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 wildcards 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.



Note: Find out more about CEL in the Katana Developer Guide: CEL Reference.

CEL in Parameters

Parameters that contain CEL expressions are set like any other string parameter only the value of the parameter is evaluated to a CEL expression. For example create a CEL expression on a CollectionCreate node that sets to the **/root/geo** location:

```
TIME = 0
root = NodegraphAPI.GetRootNode()
collection = NodegraphAPI.CreateNode( 'CollectionCreate',\
    root)
c = collection.getParameter( 'CEL' )
c.setValue( "( (/root/geo) )", TIME )
```

For more on CEL, and collections using CEL, see Collections and CEL.

Working with Attributes

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 and OpScript.

• The AttributeSet node is used to create, override, or delete attributes at one or more locations.

- The OpScript node is a Lua-based interface to the Op API. 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.

To learn how to manipulate attributes, refer to either:

- AttributeSet Nodes
- OpScript Nodes
- Working with Group Attributes

AttributeSet 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**.

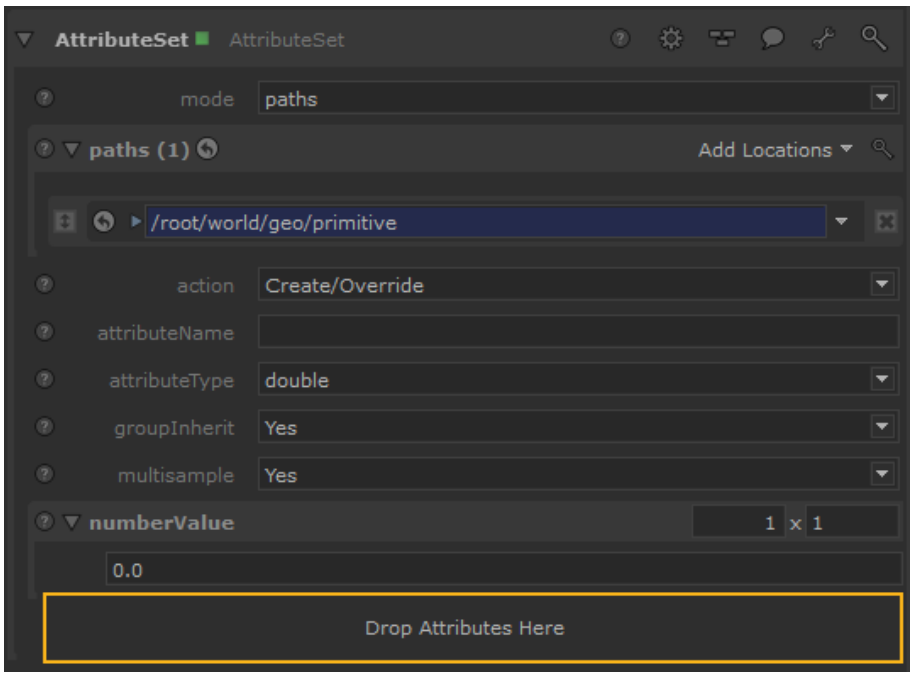
   You can enter a grouped attribute by separating the parts of the attribute with a **.** (period), for instance **geometry.point.P**.

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. Use the **multisample** parameter to select whether you want to enable multi-sampling.
10. 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.



Using Material Underlays with the AttributeSet Node

In addition to editing attributes with the AttributeSet node, as described above, you can use the node to create material underlays. Attributes from an **underlayAttrs** group attribute that is part of a **material** group attribute are copied to the top level of a location's attributes during the MaterialResolve step.

This only happens if matching attributes are not already set on the target location (the location that points at the material using **materialAssign**). This allows, for example, custom renderer object settings to be specified for locations, which can be overridden by any locally-set values.

To set a material underlay:

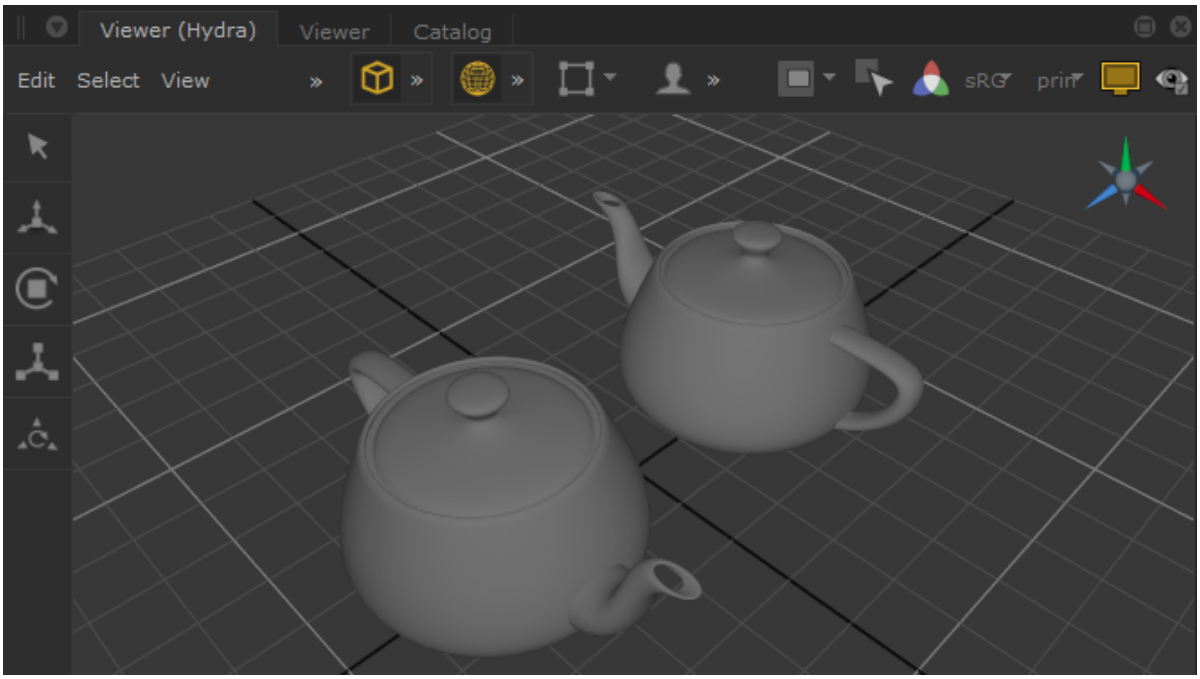
1. Add a PrimitiveCreate node and set its **name** to **/root/world/geo/teapotYellow.**

Set **type** to **teapot** and Z **translate** to **-2.5,** and Y **rotation** to **180**.

2. Add another PrimitiveCreate node (or copy and paste the exisiting one) and set its **name** to **/root/world/geo/teapotPink.**

   Set **type** to **teapot**, Z **translate** to **2.5.**

3. In the **Node Graph**, select both Primitive Create nodes and press **M** to merge.
4. Expand the **Scene Graph** to view the primitives. The image below shows the scene in the Hydra Viewer.



5. Add a yellow material node:

   Add a Material node.

   In the **Parameters** tab, change it's **name** to **Yellow**.

   Click **Add Shader** and pick **dl** > **Surface** to add a **dlSurfaceShader**. From the dropdown select **dl3DelightMaterial**.

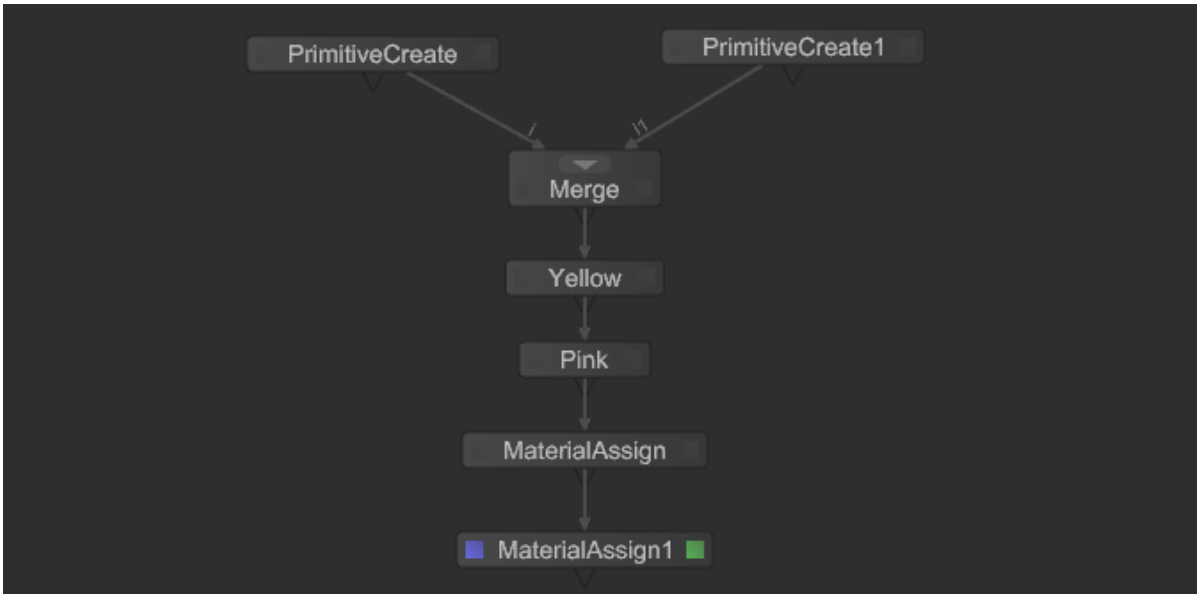
   Change the **Base Layer** > **Diffuse and Subsurface** > **Color** value to **0.5**, **0.5**, **0.0.**

6. Add a pink material node: Repeat step 5 but change the name to **Pink** and the color to **0.5**, **0.0**, **0.5**.
7. Add a MaterialAssign node. Assign the yellow material to the yellow teapot:

   Middle-mouse +**drag** the **teapotYellow** mesh item from the **Scene Graph** tab to the **CEL** field's **Add Statements** button on the Parameters tab for the **MaterialAssign** node.

   Middle-mouse+**drag** the **Yellow** material from the **Scene Graph** to the **materialAssign** field.

8. With a new MaterialAssign node repeat step 9, but assign the **Pink** material to the **teapotPink** mesh.
9. Your node graph should be connected as follows:



10. Add an AttributeSet node and connect it to the bottom of the graph. **Middle-mouse**+**drag** the **Yellow** material from the Scene Graph on to the CEL path field and set the following parameters:

    action: Create/Override

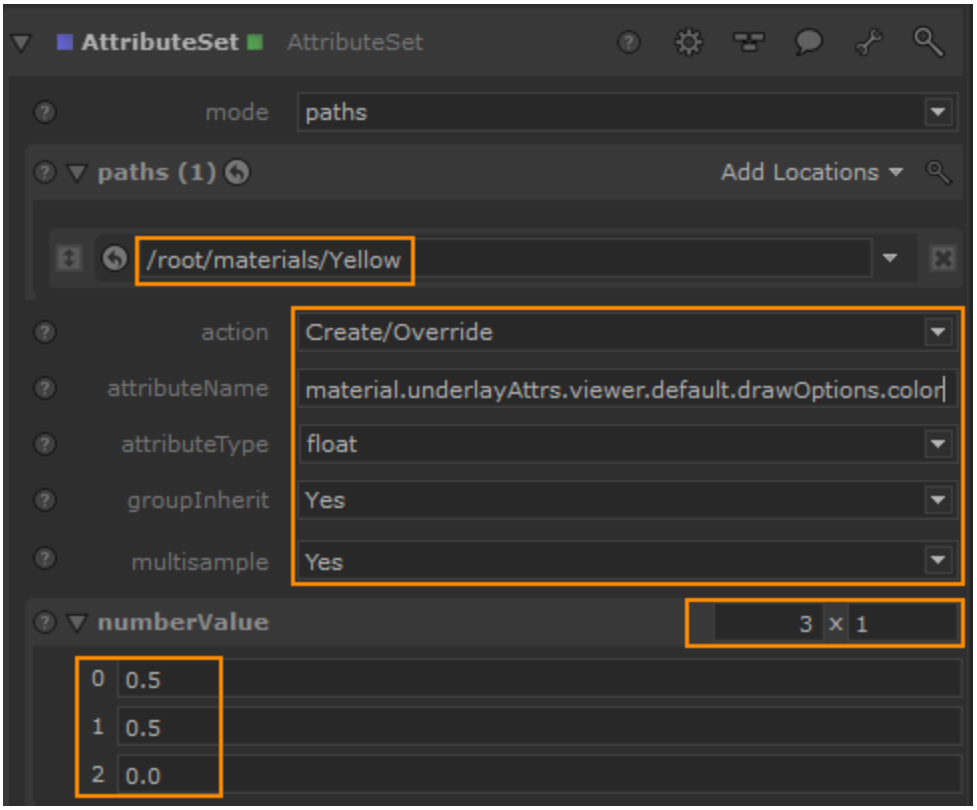
    attributeName: material.underlayAttrs.viewer.default.drawOptions.color

    attributeType: float

    groupInherit: Yes

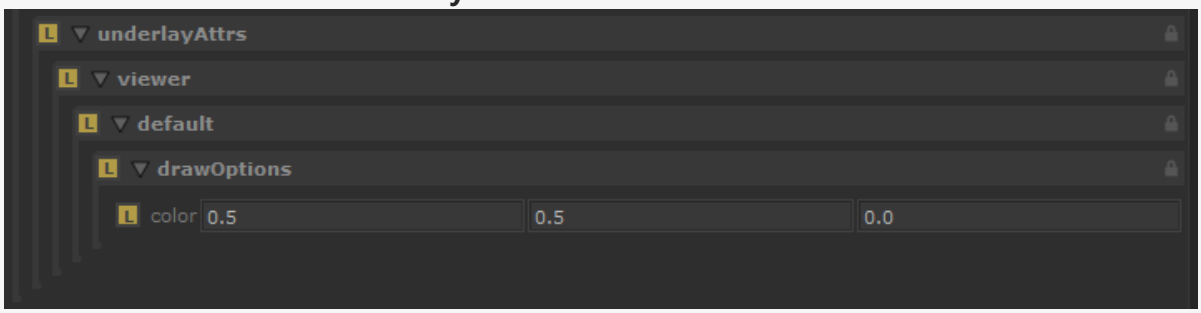
    multisample: Yes

    numberValue: 3 x 1, with **0**: 0.5, **1**: 0.5 and **2**: 0.0.



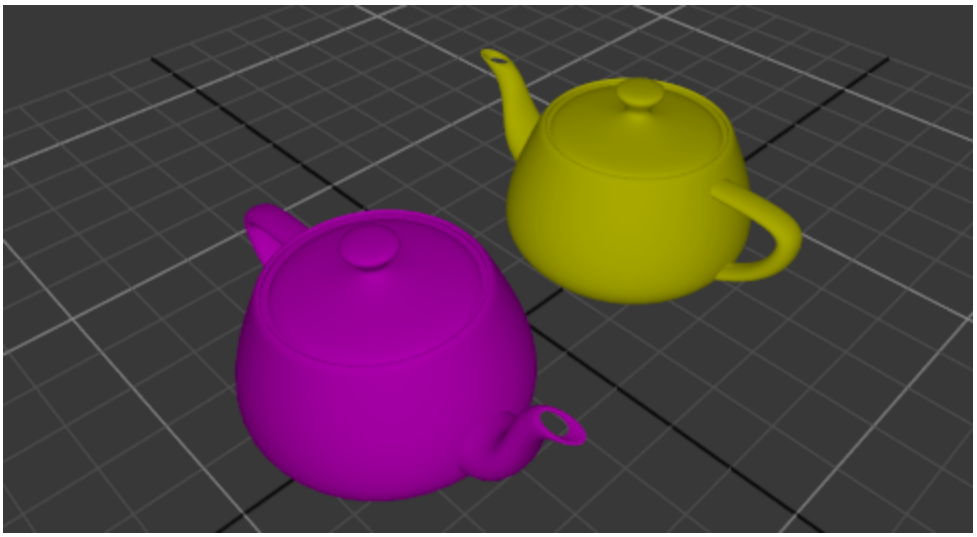
11. Copy and paste the **AtrributeSet** node and connect it to the previous node. On this new node assign the Pink material to the path and set the number values to 0.5, 0.0 and 0.5 respectively.

Note: If you inspect the materials in the **Attributes** tab, you will notice an attribute has been created under a tab called **underlayAttrs** for both materials.



Inspecting both teapot mesh items will show their viewer.default.drawOptions attributes at the default values.

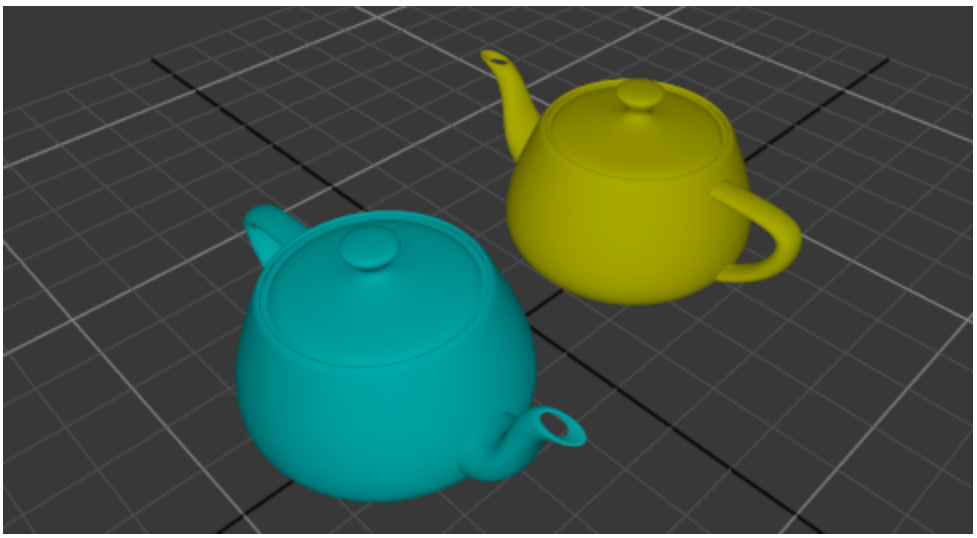
12. Add a MaterialResolve node and connect it to the previous node. The underlay attributes are transferred to the teapots and override their default Hydra view.



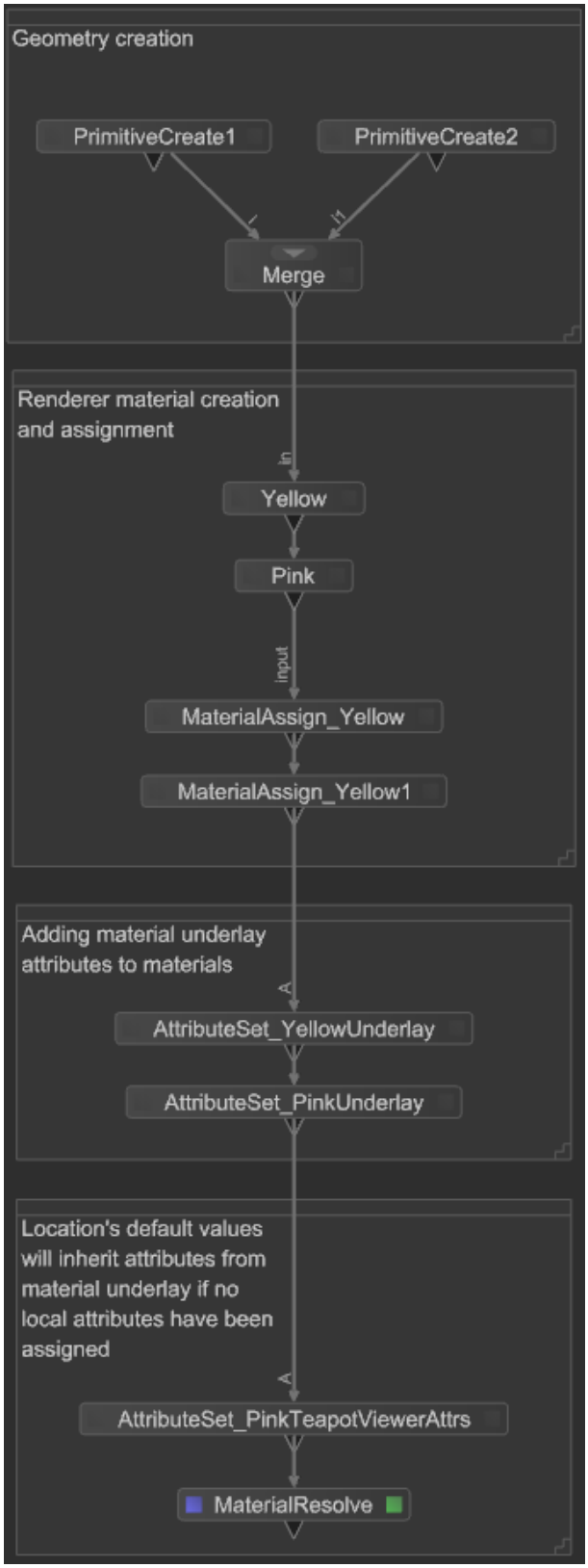
To deactivate the node and return to the default, highlight it and press **D**.

13. Copy and paste the second AttributeSet node, and connect the newly created node in between the second AttributeSet node (for the pink material) and the **MaterialResolve** node.
14. In the new AttributeSet node, change the CEL field to point at the **teapotPink** location rather than the pink material. In the **attributeName** field, change the value to viewer.default.drawOptions.color, and the **numberValue** to 0.0, 0.5, 0.5.

    You will now see the material underlay attribute is only affecting the yellow teapot at resolve time.



Here's the full node graph for reference with nodes renamed for clarity:


Geometry creation
PrimitiveCreate1
PrimitiveCreate2
Merge
Renderer material creation and assignment
Yellow
Pink
input
MaterialAssign_Yellow
MaterialAssign_Yellow1
Adding material underlay attributes to materials
AttributeSet_YellowUnderlay
AttributeSet_PinkUnderlay
Location's default values will inherit attributes from material underlay if no local attributes have been assigned
AttributeSet_PinkTeapotViewerAttrs
MaterialResolve

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 (Collection 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.scenegraphLocationModifers** group attribute.
   - **during katana look file resolve** - the script and its arguments are added as attributes under the Scene GraphLocationModifers 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. See the Reference Guide section for more information.

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**.

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 OpScript 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

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.

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 OpScript 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.

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.
```



Note: See more in the Katana Developer Guide: Attributes (C++) - GroupBuilder.

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.
```



Note: See more in the Katana Developer Guide: Attibutes (Python) - GroupBuilder.

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.
```



Note: See more in the Katana Developer Guide: Attibutes (OpScript) - GroupBuilder.

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.)

Inheritance Rules for Attributes

By default, attributes are inherited from parent locations. However, attributes can be overwritten at specified locations where the values differ from ones defined higher up in the hierarchy, as used for Light Linking.

Some attributes are not inherited, for instance the globalStatements of a renderer defined at /root or the globals defined at /root/world. Another example is the xform attribute, where it would not make sense to inherit a transform defined for a group to all its children and thus perform the operation multiple times.

Setting Group Inheritance using the API

To prevent an attribute from being inherited, use the API function **setGroupInherit()** to disable group inheritance. For example:

```
FnKat::GroupBuilder gb;
gb.setGroupInherit(false);
gb.build();
```

USD in Katana

Katana 7 and above has seen further enhancements to support the industry adoption of USD. Artists can take advantage of USD's fast and efficient scene management capabilities while still utilizing Katana's interactive rendering, flexible shading system, and advanced lighting techniques.

Katana’s USD Support

Katana started its USD integration journey by incorporating Pixar's **UsdIn** suite, which allows artists to import USD into Katana. We introduced the **UsdMaterialBake** node, offering a versatile framework for **exporting materials and lights**.

As this has progressed, we have introduced more **Native USD nodes** while ensuring that Katana’s new USD-native framework can live alongside Katana's pre-existing architecture. This avoids having to uproot entire workflows and provide a safety net as users progressively adopt USD.

Depending on how you integrate USD into your workflow, there are three sections of the user guide to refer to:

1. Using Native USD Workflows - this section is useful for USD Native workflows in Katana that is based on the new framework built specifically for USD data.

2. Bridging between Katana Geolib and USD - this section is useful for partial or progressive usage of USD alongside Katana Geolib data and assets.

3. Pre-existing UsdIn Workflows - this section is useful for workflow examples using pre-existing solutions such as UsdIn, which uses Katana’s proprietary data (Geolib).

See further explanations about these different workflows at USD Workflows - Choose your route.

USD User Guide pages:

USD in Katana

What is USD?

USD Workflows - Choose your route



Using Native USD Workflows

Building USD Scenes

Composition Arcs - Understanding LIVRPS

Importing USD Data

Native USD Prims

Native USD Lights

Creating Native USD Cameras

UsdMaterialAssign Workflows

Inspecting USD Properties Using the Attributes Tab

Manipulating USD Data Natively

Native USD Transformations

Bounding Boxes in USD Scenes

Pattern-Based Collections with USD

Native USD Layer Export

USD Debugging

USD Text View

USD Error Handling

Using the Scene Explorer

Payloads in the Scene Explorer

Activating and Deactivating Prims

Collections in the Scene Explorer

Bridging between Katana Geolib and USD

Conversion Nodes - KatanaToUsd/ UsdToKatana

Pre-existing UsdIn Workflows

Loading in USD Assets with UsdIn

Setting up USD Lights

Setting up UsdPreviewSurface Materials

Loading USD Plug-ins into Katana

What is USD?

USD, or Universal Scene Description, is an open-source framework developed by Pixar Animation Studios. It is engineered for scalability and is designed to efficiently manage and describe large-scale 3D scenes and assets. USD enables pipelines to work collaboratively with assets and scenes, assembling, disassembling, reading, writing, editing, and exchanging complex 3D content between various applications seamlessly. It provides robust tools for referencing, layering, and composing scenes, ensuring both versatility and collaborative efficiency in production workflows.



Note: To learn more, see OpenUSD's Intro to USD.

How USD Helps Lighting and Look Development

USD offers numerous advantages to VFX and animation studios, improving both collaboration and efficiency within their pipelines.

- **Streamlined Asset Management** - USD ensures consistent asset representation, simplifying shader, texture, and light rig management across various look dev and lighting tools. With many industry-standard tools adopting USD, such as Maya, Houdini, Mari, and Nuke, it becomes much easier to exchange 3D data between different applications and pipelines.

- **Efficient Collaboration** -The layering system also offers a neat way of dividing areas of responsibility across the pipeline. Multiple artists can work on the same scene simultaneously, with one working on one aspect (or area of the scene) while another focuses on a different task. When complete, the results can be published as layers into the same USD stage.

  USD's composition engine ensures that all changes are efficiently combined. This effectively eliminates the "waterfall" approach to production, where departments are dependent on the completion of work at the previous stage in the pipeline.

- **Viewport Improvements**- The Hydra rendering architecture alongside USD offers high-quality, real-time previews and supports various render engines. This enables look dev and lighting artists to receive instant feedback, making the iterative process more efficient.

- **Variants** - USD supports the concept of variants, which let you maintain multiple alternative representations or configurations of assets. They offer an organized way to dynamically switch between versions of the same asset, enabling flexibility in production. Variants are grouped into sets of related assets, allowing you to easily pick from the set. For example, you might have multiple clothing options for a character, or different levels of detail for a mesh to tailor complexity at different stages of the workflow.

- **Scalability** - USD is designed to handle complex scenes efficiently. This ensures that even when dealing with sophisticated lighting setups or intricate shaders, performance remains optimized.

- **Asset Sharing and Interchangability** - USD makes it easier to share assets between departments. A look dev artist can create a shader and, using USD, that shader can be consistently applied and viewed by artists in other departments, such as animation or layout.

- **Payloads and Deferred Loading** - Payloads enable artists to load only the required parts of the scene for their work. For example, a lighting artist can delay loading geometry, attributes, or layers until they are necessary. This reduces performance and memory usage when dealing with heavy assets.

- **Rich Metadata** - USD allows for storing metadata, which can be used for asset tracking, version history, automation tasks, and other important details about shading or lighting decisions.

- **Proceduralism** - Given USD's compatibility with procedural systems, it can facilitate dynamic changes. For example, adjusting a procedural shader parameter can change the look across multiple assets.

In general, USD's innovative approach to 3D graphics data interchange brings significant value to VFX and animation studios. It enables enhanced collaboration, non-destructive editing, a standardized format, asset reusability, and eliminates the restrictive waterfall workflow. USD revolutionizes every studio's workflows, leading to more efficient pipelines.



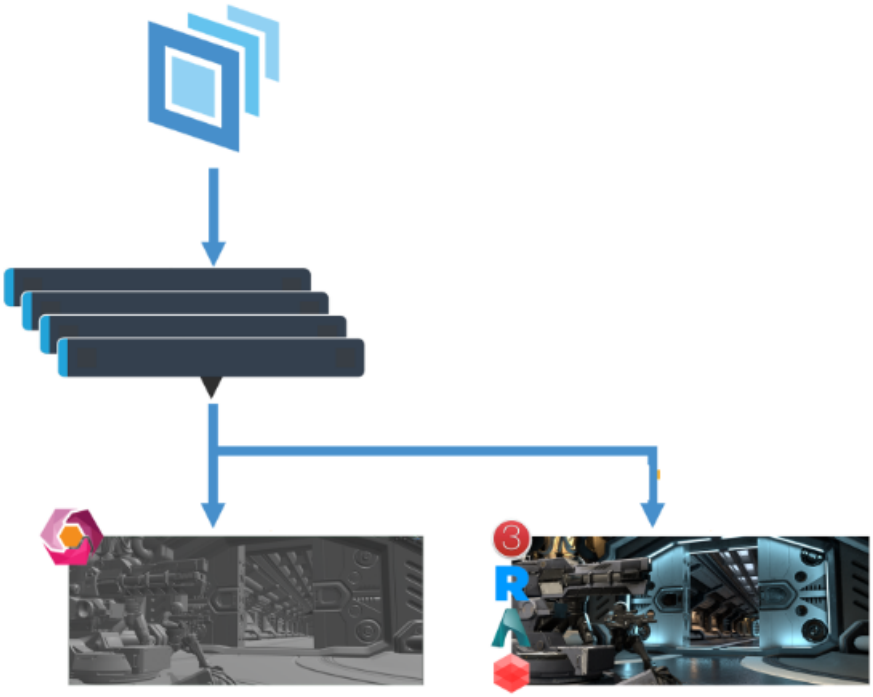
Note: To start learning how to build a USD scene, see Building USD Scenes.



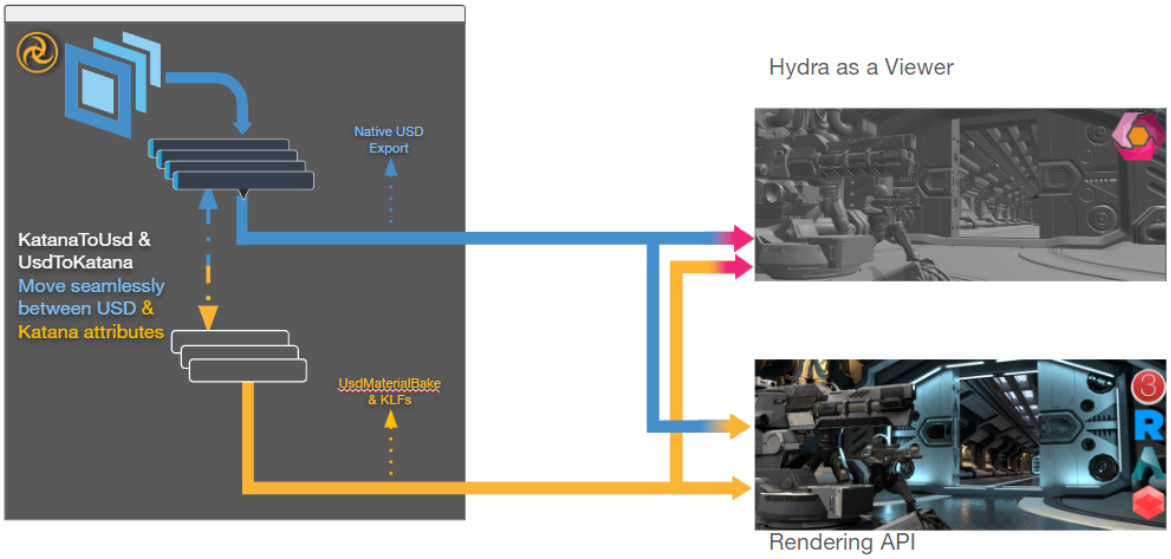
Note: To see more about USD workflows in Katana, see USD Workflows - Choose your route.

USD Workflows - Choose your route

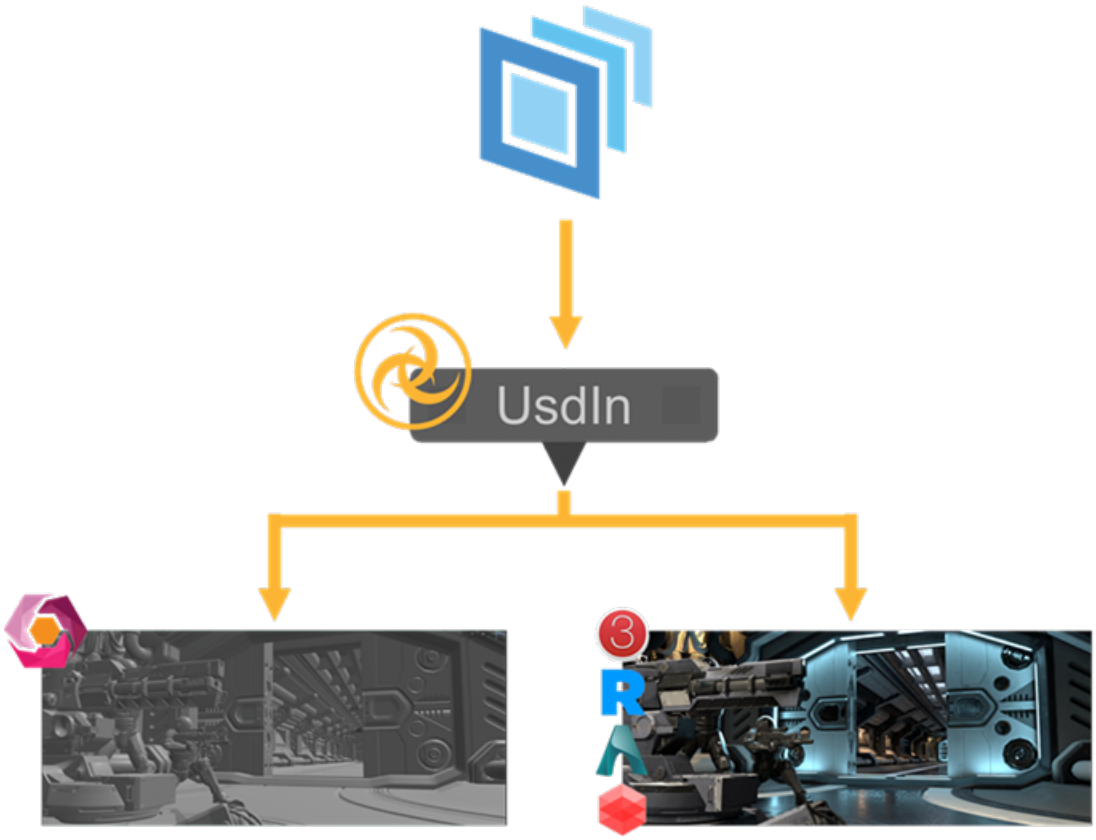
1. "Using Native USD Workflows" allow you to complete your workflow while staying in USD space. Your USD data and assets are created, edited and rendered seamlessly. We are continuing to work on Native USD to enable more options for these workflows.



2. "Bridging between Katana Geolib and USD" refers to the ways in which the new USD-native infrastructure can be used in conjunction of any other features and workflows in Katana. These processes allow you to work in a hybrid way where you can mix both workflows whilst in the process of adopting USD.



3. "Pre-existing UsdIn Workflows" refers to USD workflows that have existed since Katana 4.0, prior to there being a native USD framework. In recent versions, this is still a valid workflow which undergoes an immediate conversion to Katana's Geolib data; its USD-specific properties can be edited using nodes that begin with "UsdIn", however the data is not USD-native. Using these methods means that your workflow will be predominantly using a combination of UsdIn with nodes that manipulate Katana attributes.



Using Native USD Workflows

New Nodes for Native USD Stage Creation and Manipulation

We introduced new nodes coupled with an underlying framework that allows for native USD manipulation.

Composition	**Prims**
• UsdInheritSet	• UsdCamera
• UsdPayloadSet	• UsdLight

• UsdReferenceSet • UsdSpecializeSet • UsdSubLayerAdd	• UsdPrimCreate • UsdSchemaSet • UsdScope • UsdXform
Properties • UsdActiveSet • UsdAttributeSet • UsdCollection • UsdMaterialAssign • UsdMetadataSet • UsdKindSet • UsdPrimvarSet • UsdRelationshipSet • UsdStageDefine • UsdTransformSet • UsdTransformEdit • UsdVariantSet	**Output** • UsdLayerDefine • UsdLayerExport • UsdLayerExportGroup
Other • UsdLayerWrite • UsdPythonWrite • KatanaToUsd • UsdToKatana	



Note: A full set of links to the reference guide topics for each node can be found in USD Nodes.

The nodes are classified according to the following types:

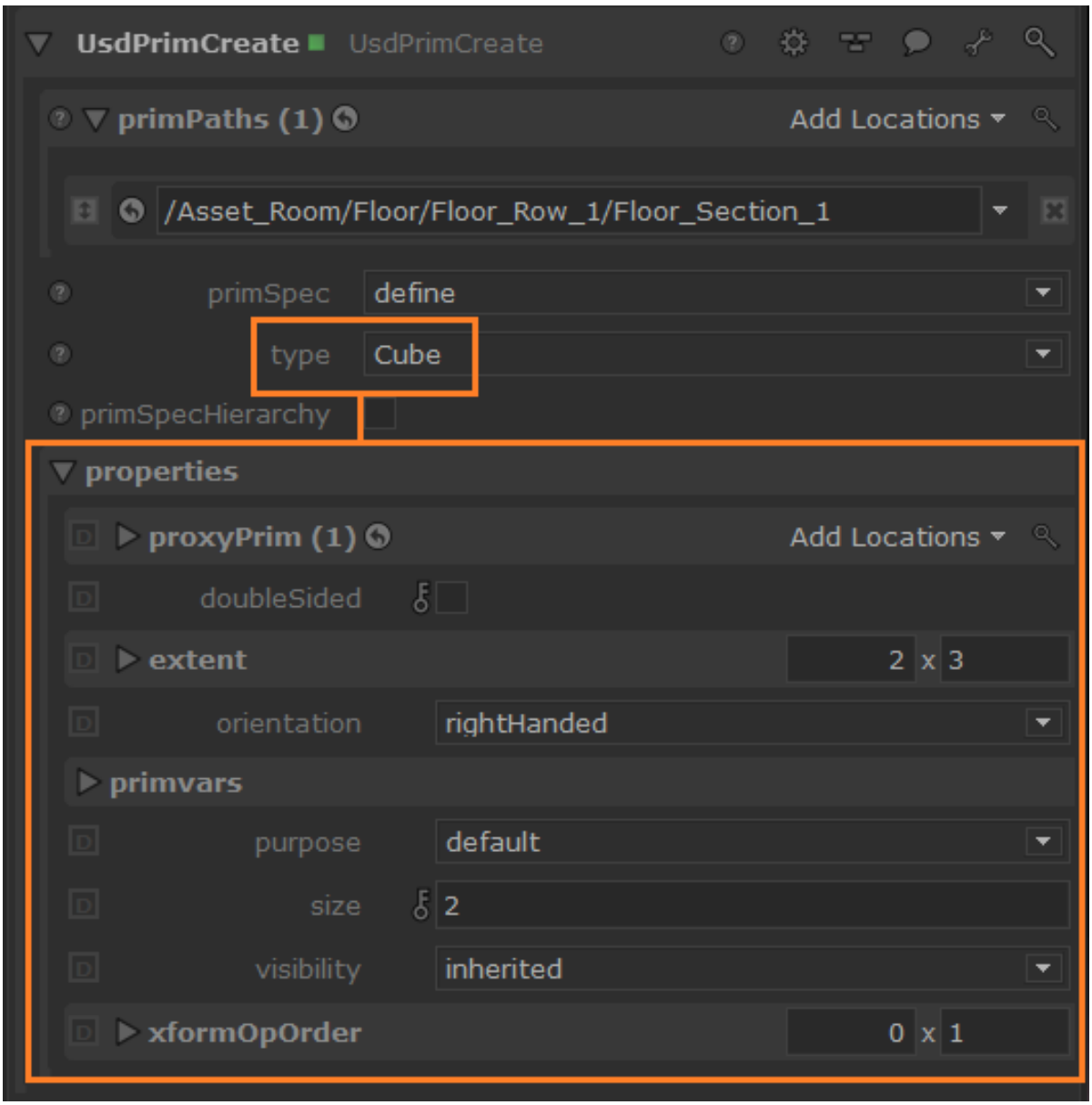
- **Composition** - New nodes for scene composition are determined by LIVRPS ordering. These are guidelines ordered by Local, Inherits, VariantSets, References, Payload, and Specializes that determine how pieces of a 3D scene work together effectively. For example, UsdSubLayerAdd to add a layer to the stage.

  See Composition Arcs - Understanding LIVRPS for more information about this.
  See USD Nodes for links to the reference guide topics.



Note: In Katana 7, VariantSets are not included for authoring but you can change which variant is used via the UsdVariantSet node.

- **Prims** - The following nodes to let you create and modify schemas. UsdPrimCreate for prim generation and overriding, and UsdSchemaSet to assign a schema API to a prim.
  For both nodes, Katana speeds up the workflow by tailoring the UI to the selected prim or schema type automatically. Simply select a type, and the node properties are dynamically populated by the USD version used, with its parameters defined within the USD API itself.



A cube type with matching parameters generated dynamically.

- Additionally, there are more specialised nodes, such as UsdLight, UsdCamera, UsdScope and UsdXform nodes, based on the UsdPrimCreate but with parameter tailored for each purpose, allowing you to create different types of prims easily.
  See Native USD Prims, Native USD Lights, and Creating Native USD Cameras.

- **Properties** - New nodes for essential stage modifications. For example, UsdTransformSet and UsdTransformEdit to apply a transform to a prim, and to enable viewer manipulators for native USD. See Native USD Transformations.
  USD's Pattern-Based Collections (PBC) is also integrated into Katana. Similar to Katana's CEL, the PBC widget and UsdCollection node allows you to use, define and modify USD collections, helping make workflows more efficient. See Pattern-Based Collections with USD.

  For more about assigning materials in USD, see UsdMaterialAssign Workflows.

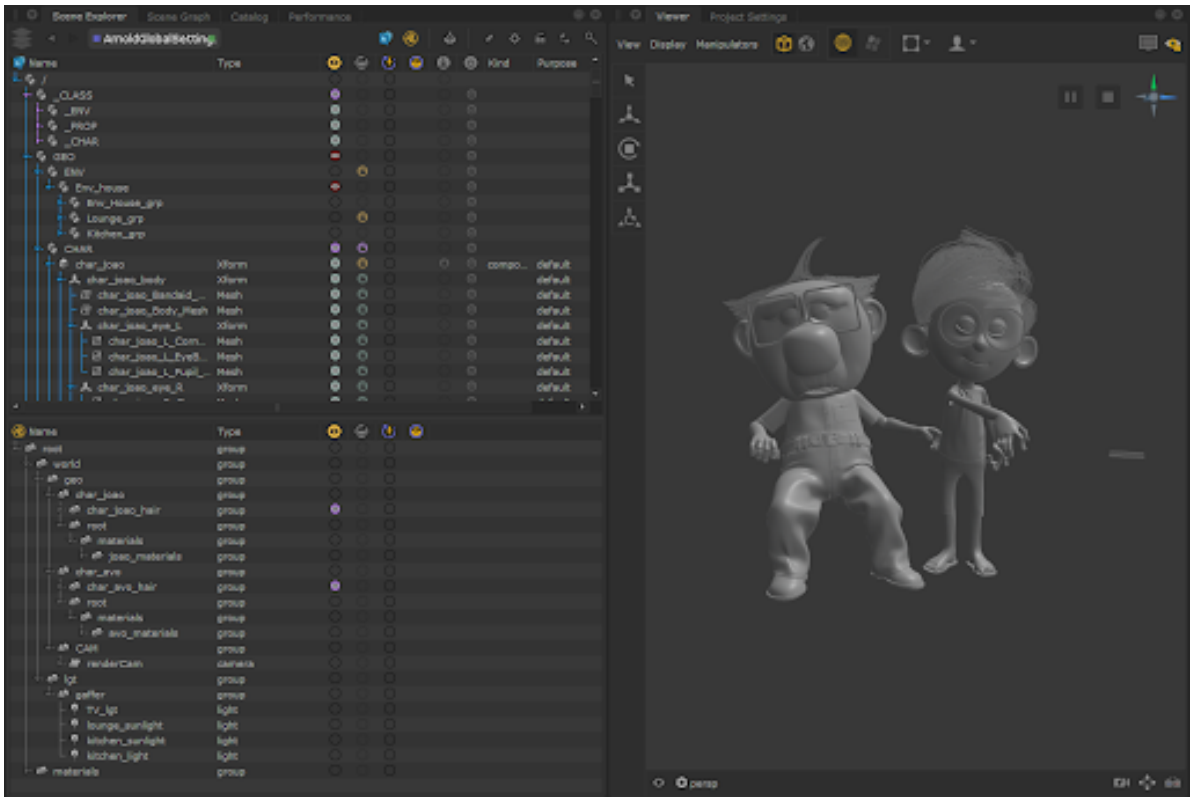
  See USD Nodes for links to the reference guide topics.

- **Output** - You can now use UsdLayerExport and UsdLayerDefine to bake native USD layers. See Native USD Layer Export.

- **Other** (Pipeline/TD Tools) - We've added a UsdLayerWrite node so that you can take or create a scene description stored as .usda, edit it, and manually write it to the stage. This is useful when you need to quickly add content to locations in the scene. Plus, UsdPythonWrite gives you the power of Python to author USD layers. To make workflows more flexible, UsdToKatana and KatanaToUsd let you convert USD data into Katana data, and vice versa. You also can use USD Text View to view .usda data based on the selected node, helping with debugging.

Scene Explorer

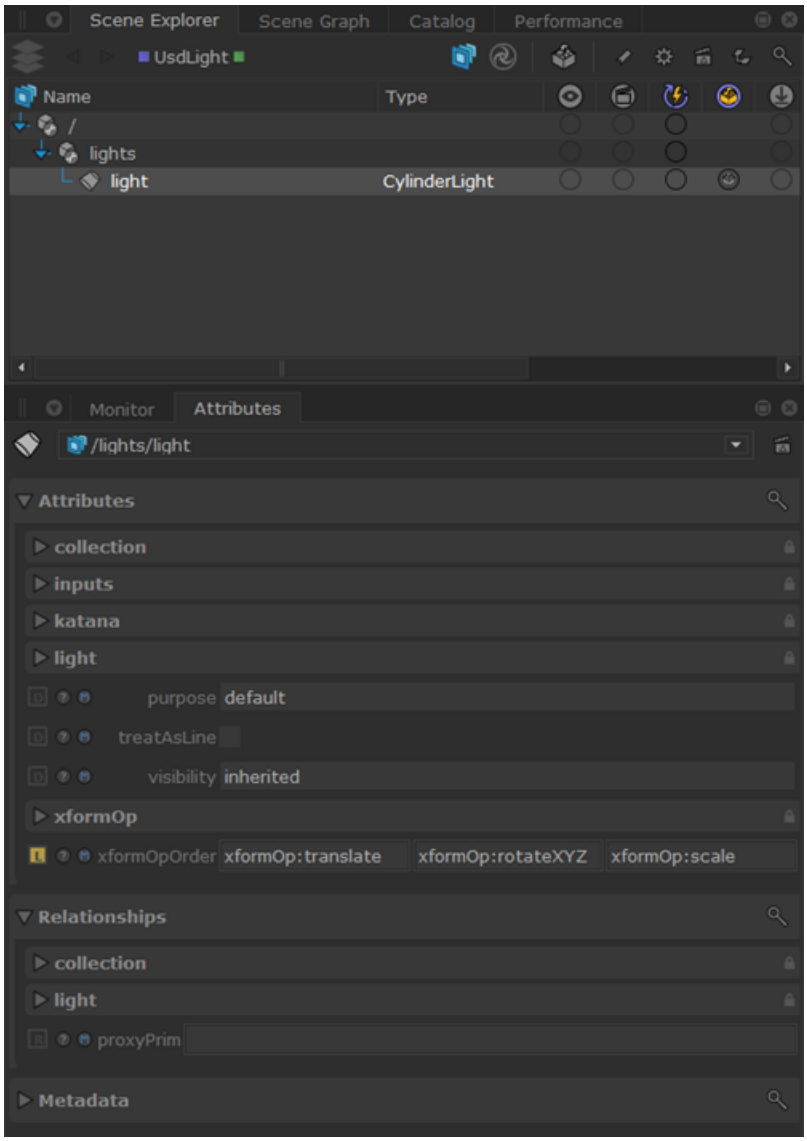
The **Scene Explorer** tab provides a unified scene graph view of both USD and Katana data so that you can easily traverse and manipulate the hierarchical structure of the entire asset library.



For more information see Using the Scene Explorer.

USD Properties Viewable in the Attributes Tab

We've integrated USD property inspection into the Attributes tab, making it easy to check the attributes, relationships and metadata for prims and properties in the scene. Just click on a prim in the Scene Explorer to see the data in the Attributes tab. See Inspecting USD Properties Using the Attributes Tab.

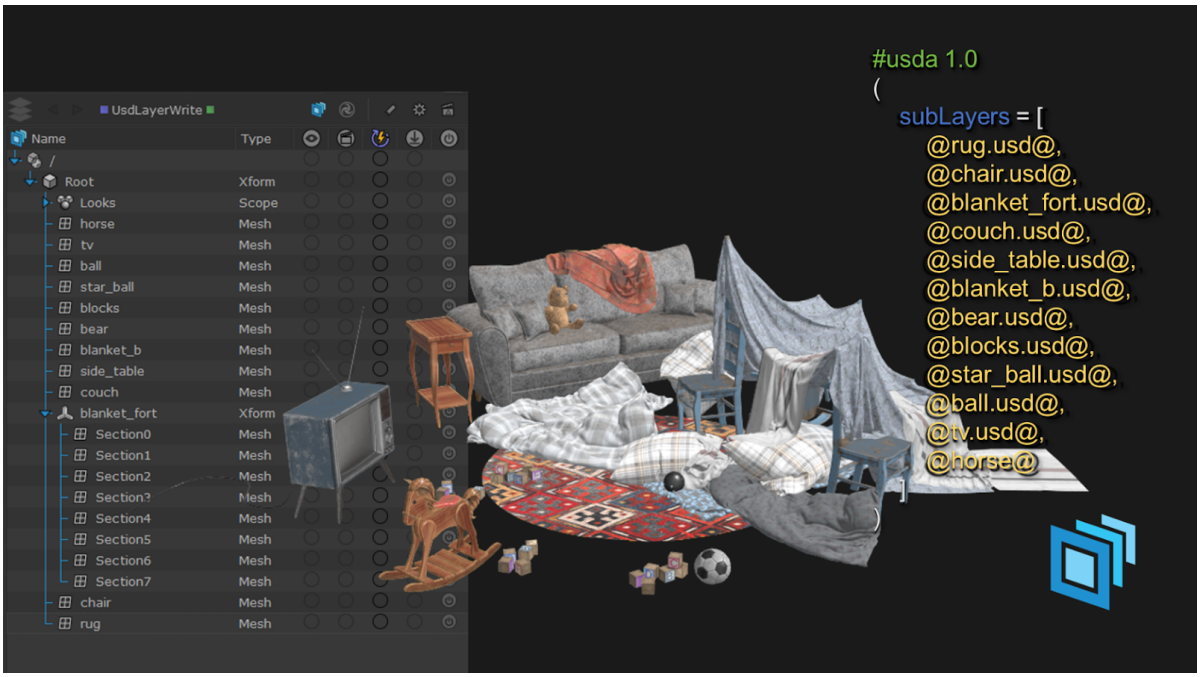


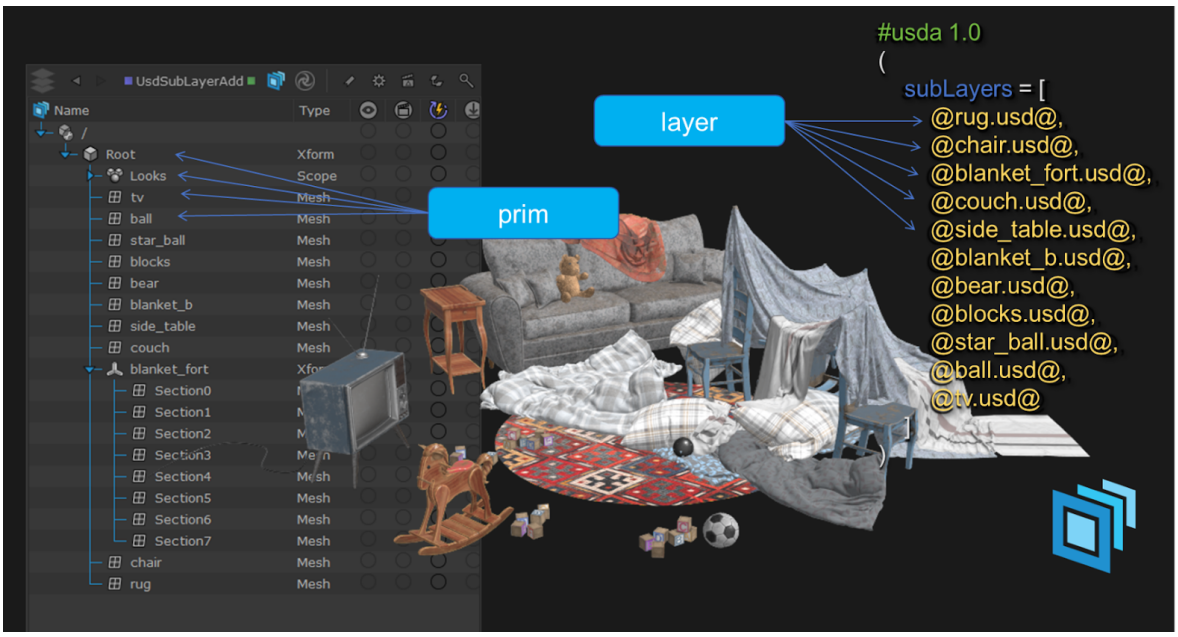
Building USD Scenes

A USD scene is composed of multiple **layers**.

A layer is a container that comprises of information about an asset (eg. file/object), properties on an asset, its relationships and more.

Here's an example of how a layer might look:





- In USD, objects and locations are called **Prims**, and come in many forms, such as cubes, meshes, and xforms.
- Prims can have various Properties which make up its **Attributes**. Editing these changes the Prim itself.
- Properties also define **Relationships**. These are the connections and interactions between Prims.

Note: To read more about USD terminology, see OpenUSD's USD Glossary.

How does it fit together?

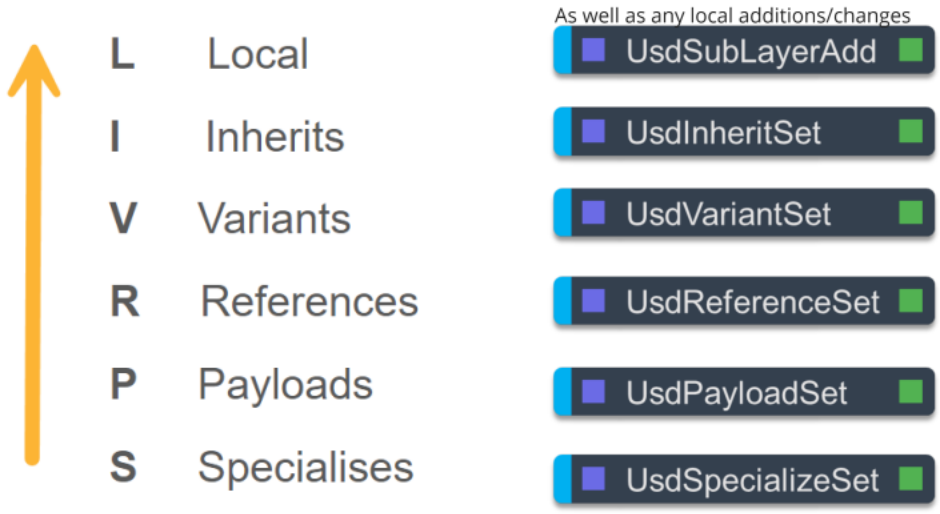
In order to put together a scene, you will need various types of layers. Different layers have different functions, and certain layers naturally override others. Understanding and composing this hierarchy is key to building a USD scene. To learn about this, see Composition Arcs - Understanding LIVRPS.

Composition Arcs - Understanding LIVRPS

In order to put together a scene, you will need to use different types of layers.

When working with USD, there is a set of rules defining which layers override others. Understanding and composing this hierarchy is key to building a scene.

There are six types of layers in USD. Layers that are “higher on the list” have stronger opinions than those lower down.



Understanding the composition rules

Get to grips with USD layers and composition arcs in this video exercise. Using simple cube prims, learn how to compose scenes in Katana with multiple layer types and see LIVRPS in action.

Tip: This project can be opened directly from Katana from **Help** > **Example Projects**> **Native USD** which you can use to follow along with the exercise or quiz yourself.

Layer Types

Local

Local describes a layer that is made locally. This means that any further changes made after you've composed your stage, will be stored as local value. They contain opinions on properties (e.g. cube size, colour etc.) and will override others.

Inherits

Inherits will take on the properties and hierarchy defined in the prim they're inheriting from. This will remain "live" and therefore will always reflect updates on its "base", as long as no further local overrides are made.

Variants

Prims can have variants, which are groups of properties that you can switch between.

References

References enable the reuse and assembly of content by incorporating the contents of one layer into another. References can be versioned, and allow local changes to be made, without impacting the original file.

Payloads

Payloads are similar to references. A heavy or complex layer could be brought in as a payload in order to avoid processing and improve performance.

Specializes

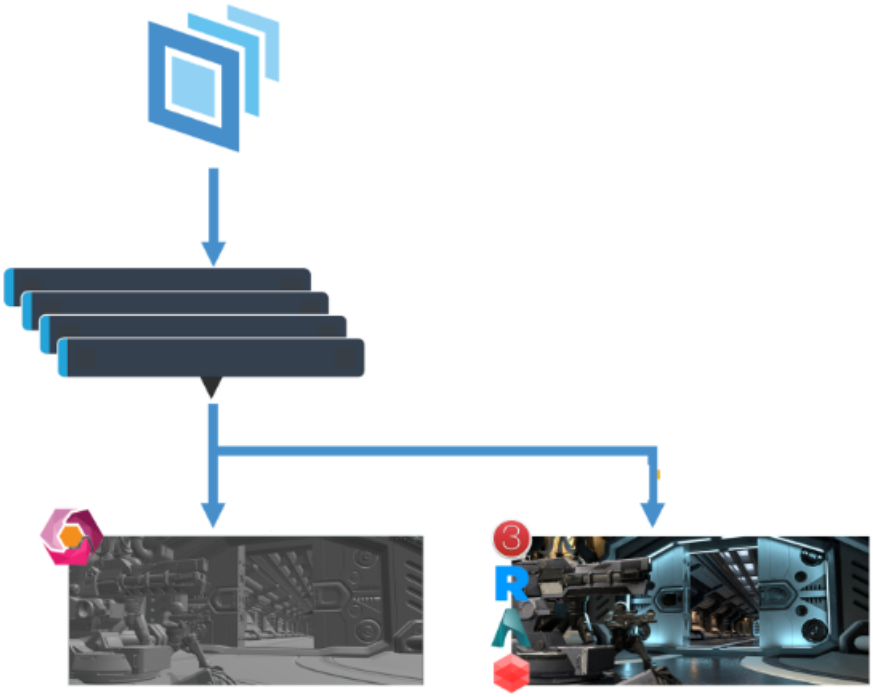
A specialize acts as the baseline values and is overridden by higher layers and arcs.



Note: For more information, see LIVRPS Strength Ordering in OpenUSD's USD Glossary.

Importing USD Data

Native USD workflows do not convert the data to Katana locations and attributes, so the data can be manipulated with native USD nodes inside Katana. This brings a lot of benefits in terms of speed and interoperability.



Note: You still have the option to bring data across to Katana using UsdToKatana conversion nodes if needed.

There are various ways of bringing in USD data natively. You can bring in entire USD stages using the UsdSubLayerAdd node, or build your stage inside Katana using the various composition arcs below:

UsdPrimCreate - Used to create hierarchy locations of various types eg. point, cube, material, xform, etc. These can be used to define a scene hierarchy before bringing in components to the locations. This node is dynamic, so the prim type dropdown will update with prim options from the version of USD being used and parameters will be created automatically by querying the USD.

UsdSubLayerAdd - A layer is one of the fundamental building blocks of the USD stage. Subsequent layers can define new prims or override prims that were already on the stage. A Layer is a single USD (usda, usd, usdc, usdz) file saved to disk. Sublayers are used to create LayerStacks, which are just as they sound - stacks of layers. They can be used to load layers to the root level without defining a prim path, this can be useful for bringing in complete stages, or layering various prims with the same hierarchy.

UsdPayloadSet - Used for loading payloads to the scene. Any data can be brought in as a payload, a common use would be an asset with no animation or transformations, for example, a t-pose character. The look and animation/layout would then be referenced on top. They can also be used to bring in large FX or particularly heavy elements because prims brought in through Payloads have the added benefit of being

chosen to be loaded or not through the Payload working set. Payloads which aren't loaded are still visible in renders - same principle as deferred loading.

UsdReferenceSet - Used to make overrides to certain properties of a prim, for example, overriding a t-pose character's transforms to the animated transforms.

UsdInheritSet - For inheriting hierarchy and Properties from one prim to another. If the prim being inherited from is updated, all prims inheriting it will also be updated. Useful for multiple instances of a prim as you'll only need to make updates once.

UsdSpecializeSet - Similar to inheritance where you can create a specialized prim from another prim. The difference being the overrides on the specialized prim level will always be stronger than any further references. This can be useful for ensuring certain aspects are never overridden.



Note: To learn about the composition arcs, see Composition Arcs - Understanding LIVRPS. For more information and examples, see OpenUSD's USD Glossary.

Tip: To see example scenes and varying examples of how assets are brought in, you can look at the example projects at **Help** > **Example Projects** > **Native USD**.

Native USD Prims

In USD, **prims** (short for primitives) are the fundamental building blocks of a scene, and come in many forms, such as cubes, meshes, and xforms. Being able to natively create these in Katana can make USD workflows more efficient.

In Katana, you can use UsdPrimCreate to create or override prims. Katana tailors the UI to the selected prim automatically. Simply select a **type**, and the node properties are dynamically populated with the matching parameters.

The **primSpec** specifier determines how the prim is treated during composition and has the following options:

- **define** - create the primitive in the scene.
- **over** - a prim that is used only to override opinions that exist in the scene.
- **class** - for defining default attributes using a class template.

You can see examples of different primSpecs in the Scene Explorer in the section Styling in the Scene Explorer.

There are also more specific nodes, such as UsdLight, UsdCamera, UsdScope and UsdXform nodes, which allow you to create different types of prims directly.

The UsdPrimCreate node

As mentioned above, the UsdPrimCreate node is extremely flexible in creating and editing prims, in the most simple of forms.



Note: For more information on how to bring USD data from external USD files, refer to Importing USD Data.

The UsdPrimCreate is a **dynamically populated** node which will query USD's API to display the parameters against a prim's type. This dynamic approach is particularly powerful in order to keep Katana USD-version-agnostic. I.e. if you're not using the same version of USD that Katana ships with, the UsdPrimCreate will be populated based on what's present in the USD version that's currently use.

There are additional prim-specific nodes that are available within Katana that are more catered to a workflow or action. These additional nodes are all derived from the UsdPrimCreate, but are populated with parameters and tools that are more tailored to the needs of that particular prim - such as UsdLight and UsdCamera nodes.

For example: Within the UsdPrimCreate's parameters, setting the **type** to **Camera** will create a camera with default values. Alternatively, the UsdCamera node will also allow you to create a camera, but the parameters presented will be tailored to what you'd need when creating a camera.



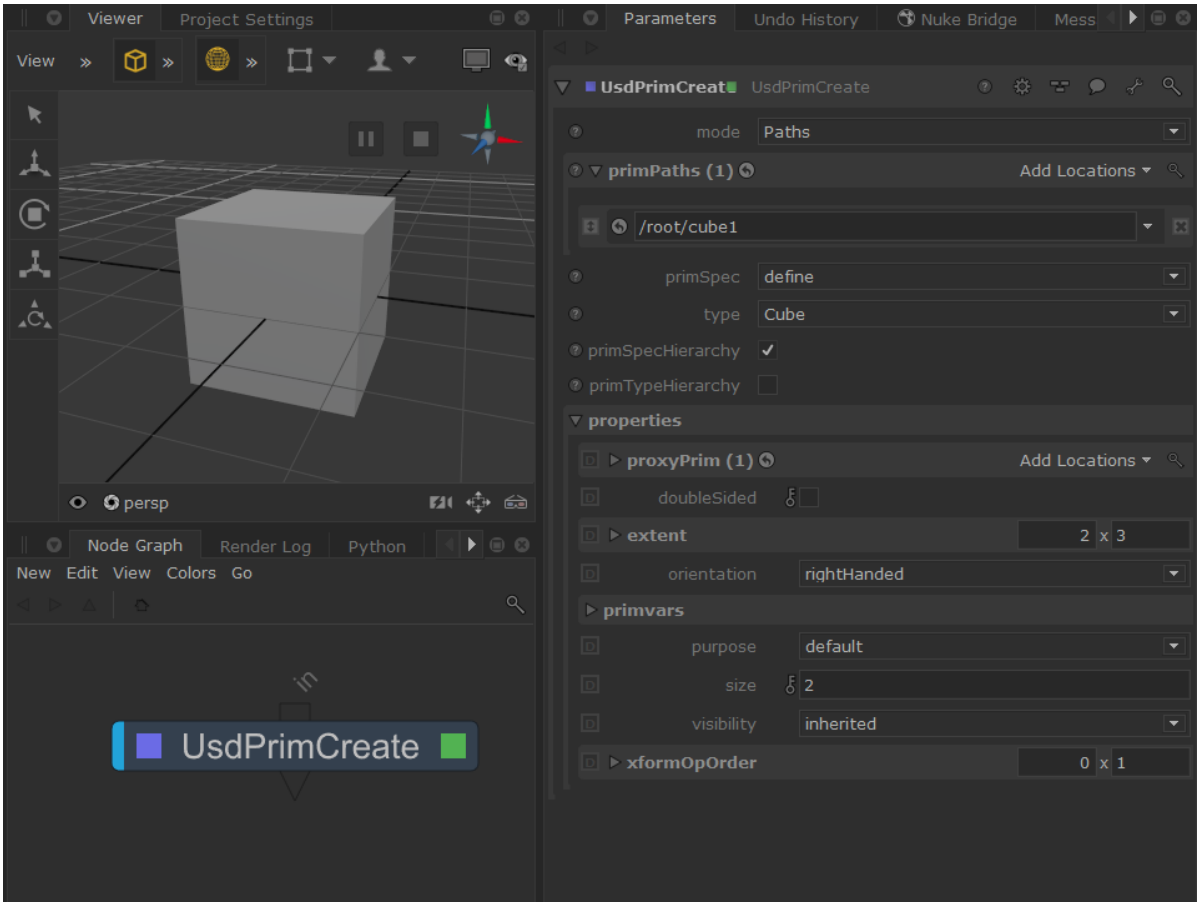
Note: For more information on these, see Native USD Lights and Creating Native USD Cameras.

Using the UsdPrimCreate node

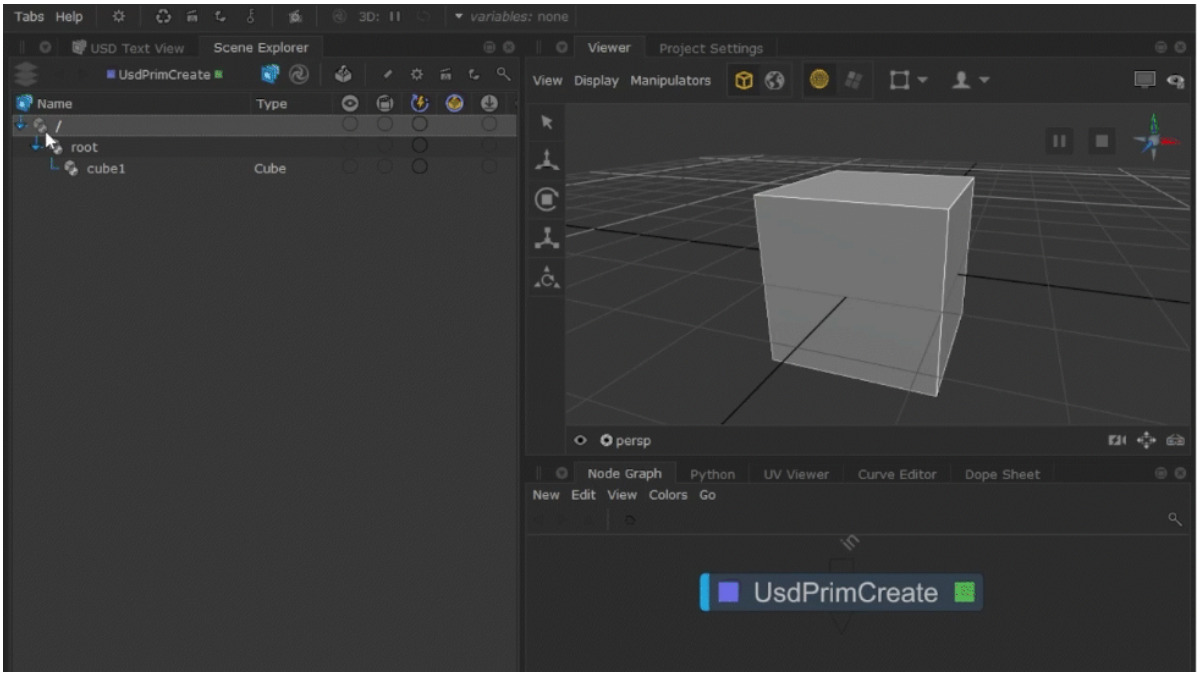
Create a Cube prim:

- In the Node Graph, hit **Tab** and select **UsdPrimCreate**.
- In its Parameters tab, set the prim's hierarchy in the Scene Explorer via the **primPath** parameter. In this example, we've set the path to **/root/cube1**.

- Set the **type** parameter to **Cube**.

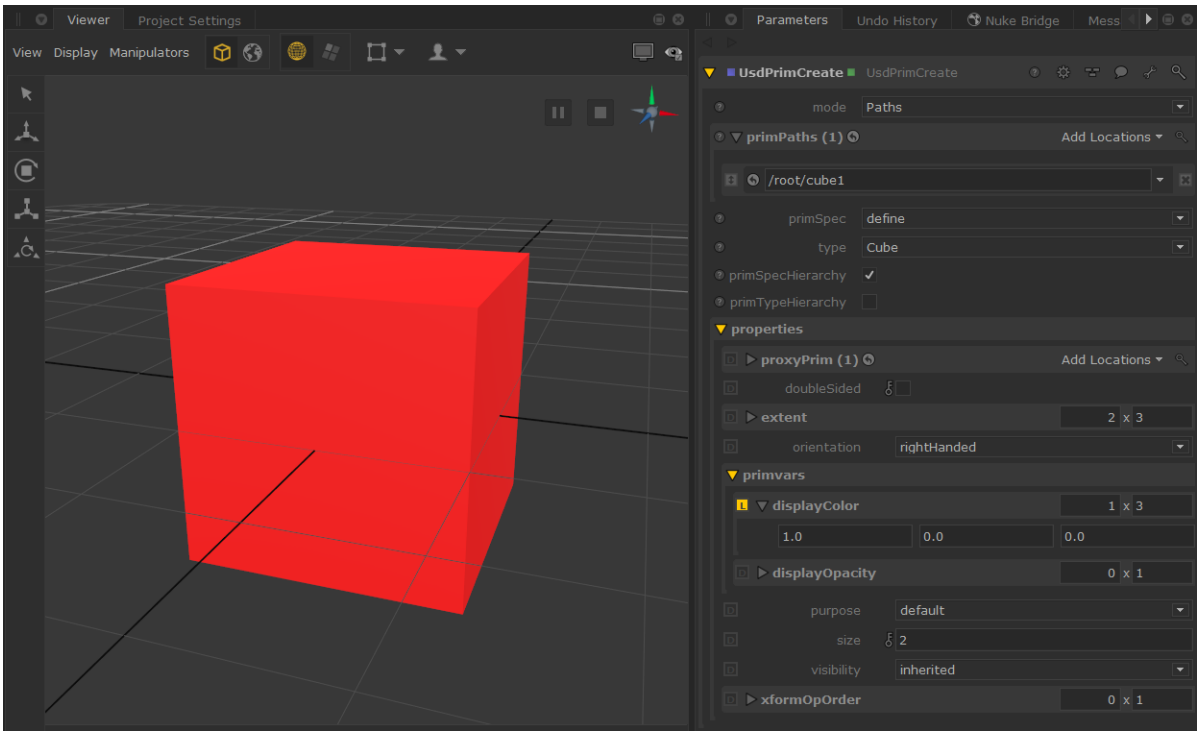


- In the **Scene Explorer** tab, expand the scene tree to the prim level, by selecting the arrows or double-click on the top prim location to expand everything.



This will result in a USD-native prim, of type **Cube**.

- This node's properties will be **dynamically populated** through the USD API, directly within the **Parameters** tab. In this example, the Cube's colour has been modified.



Note: To display the colour in the Viewer, uncheck the option in **Display** > **Basic Material**.



Note: To see the resulting usda file for any node, open the **USD Text View** tab and select the node to preview. Toggle the **Mode** parameter to display the result of the selected node, or to display all upstream results up to that node.

If you check USD Text View tab, you'll see the following usda file:

```
#usda 1.0

)

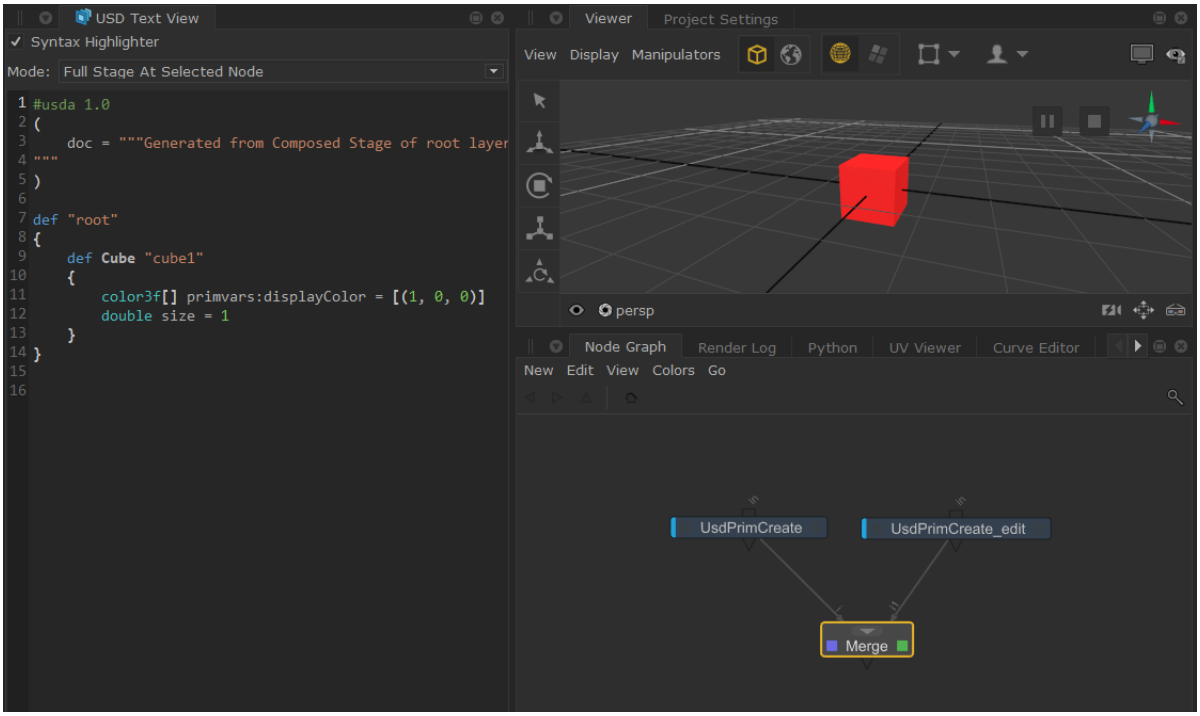
def "root"
{
    def Cube "cube1"
    {
        color3f[] primvars:displayColor = [(1, 0, 0)]
    }
}
```

Making downstream changes with UsdPrimCreate

A main rule of USD is that nothing gets "deleted" from the document. You can only deactivate elements from the scene, which will leave it unused but still present in the USD document.

By the same token, when prims of the same name and hierarchy are authored and opinions are set in multiple places, the results will record and show a combination of these opinions, as seen below:

The same is true if both of these nodes were side-by-side and merged, as seen below. This is because they're both pointing to the same prim and will combine both USD stages into one.



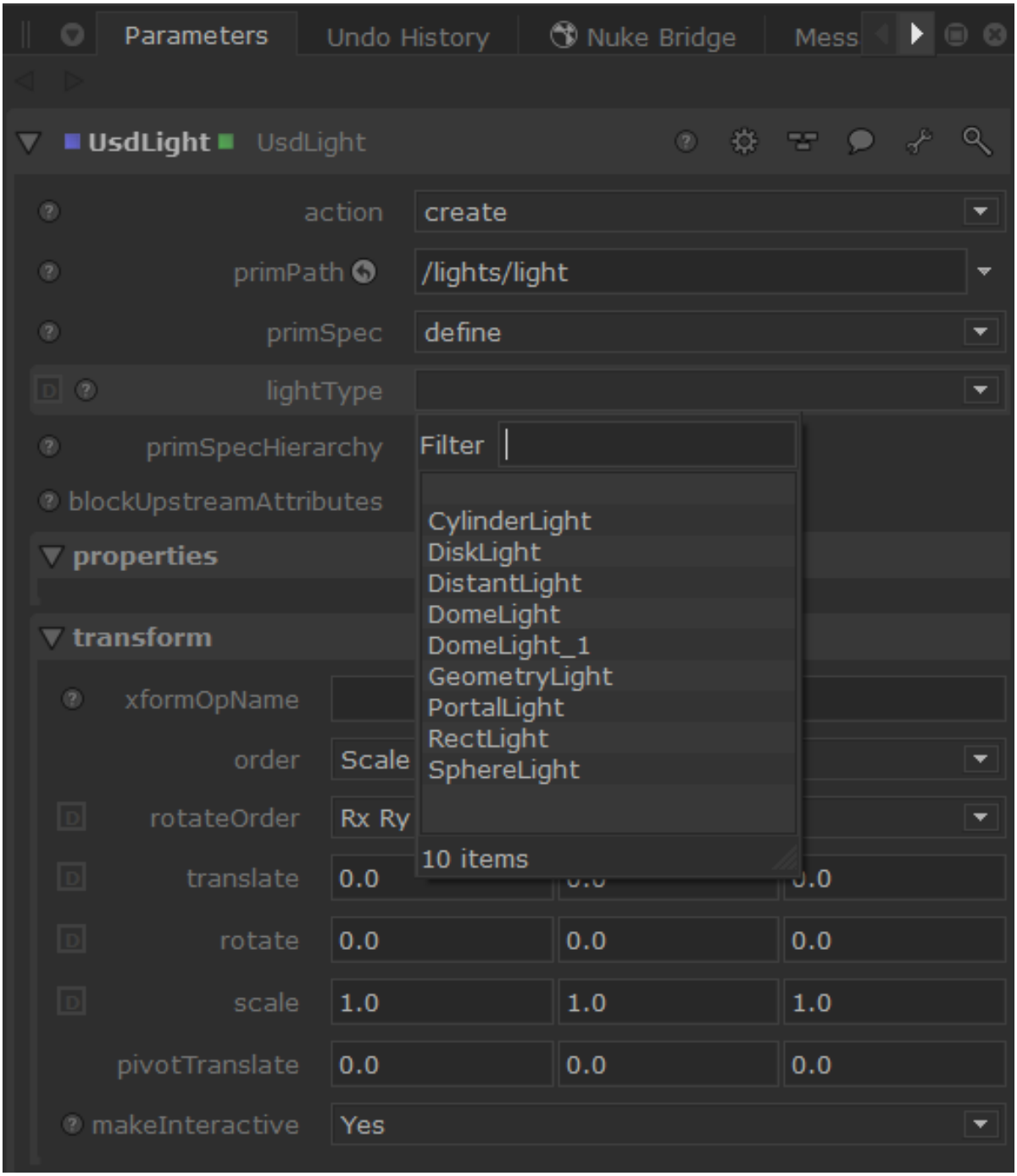
Native USD Lights

You can create native USD lights in Katana using the UsdLight node. This node also allows you to edit existing USD lights that have been imported.

Creating a Light

- Add a UsdLight node.
- In the Parameters, choose an **action** from the dropdown. To create a new light, select **create**.
- Choose a **lightType** from the dropdown. The list will include light types registered in the UsdSchemas and any render specific lights.

Basic lights you will usually find shown in the list here:



Then, adjust your light in the **properties** section:

See properties

- **Inputs**: Adjust light properties such as color, intensity and exposure.
- **Shadow**: Adjust shadow options such as the shadow color and falloff.

- **Shaping**: In the Shaping API section, you can toggle to activate and apply, as well as any parameters that come with it.
- **Texture**: Adjust texture options where relevant.
- **Extent**: Extent is a three dimensional range measuring the geometric extent of the authored gprim in its own local space (i.e. its own transform not applied), without accounting for any shader-induced displacement. Here you can categorise the geometry's purpose and visibility.
- **Light**: Define shader ID and specify the relationship of the Material response to the lighting response.
- **Transform** (xformOpOrder): Transform lights that are Xformable.



Note: This section will show different options depending on the type of light you select.

Editing a Light

You can also edit existing lights using the UsdLight node.

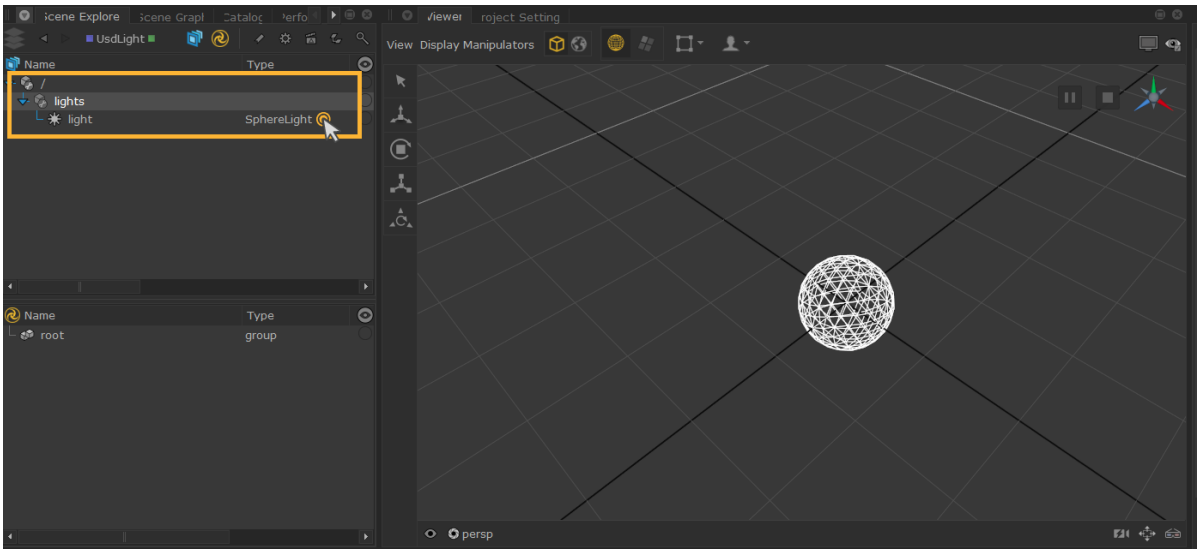
- Add a UsdLight node.
- In the Parameters, choose an **action** from the dropdown. To edit an existing light, select **edit**.
- Select your light's location in the **primPath** field - this is the light you wish to edit.
- The light **properties** will then be populated and any edits you make will override it.

USD Lights in the Viewer

USD native light locators are now visible in the **Viewer** tab.

To view them, you can simply expand the **Scene Explorer** (double-click a prim path to expand all prims below) and the light created will display in the **Viewer**.

In this example, the Scene Explorer is expanded to the "light" prim path which will reveal it in the Viewer. Since it is a SphereLight (as described in the Type column), the relevant locators will display.



Light types that this applies to

- CylinderLight
- DiskLight
- DistantLight
- DomeLight
- PortalLight
- RectLight
- SphereLight

To see practical examples of creating and editing USD lights, see Native USD Lights.

Creating Native USD Cameras

You can create native USD cameras in Katana using the UsdCamera node.

To create a camera:

- Add a UsdCamera node.
- Adjust your camera in the **properties** section:

- **Clipping Planes**: Adjust options such as clipping range, exposure and focal length.
- **Shutter**: Adjust options such as vertical aperture.
- **Transform** (xformOpOrder): Transform cameras that are Xformable.

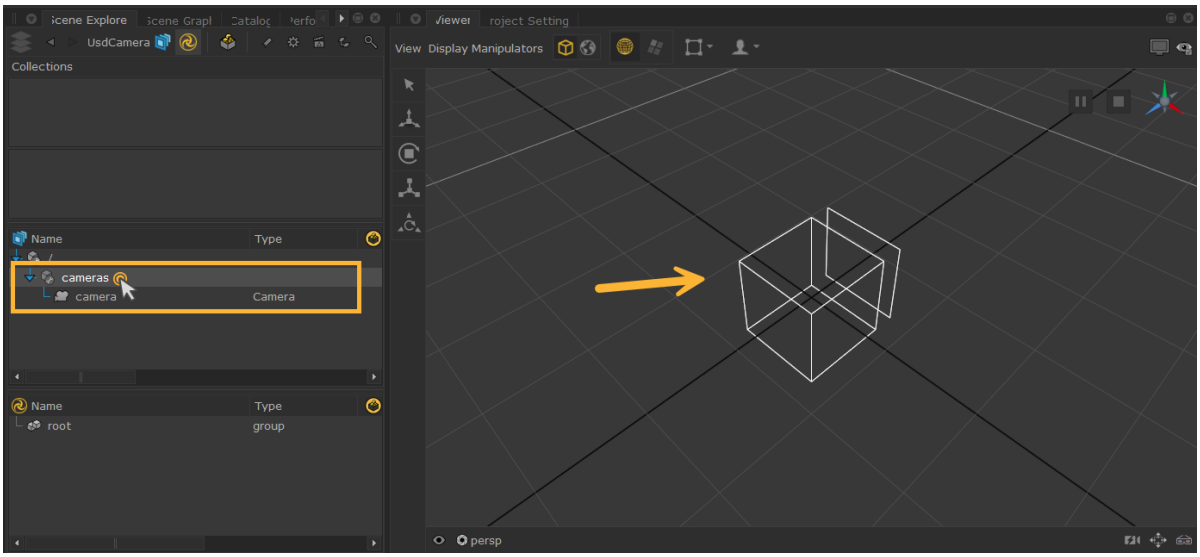
USD Cameras in the Viewer

To view a USD camera in the Viewer, simply expand the Scene Explorer to the camera prim level.



Tip: To expand all locations under a prim, simply double click on it to expand all its context to its leaf level.

In this example, double clicking 'cameras' toggles the visibility of the Camera in the Viewer:



Looking through a Camera

You can change the view to a USD camera to make placement easier or to help with composition. To set this:

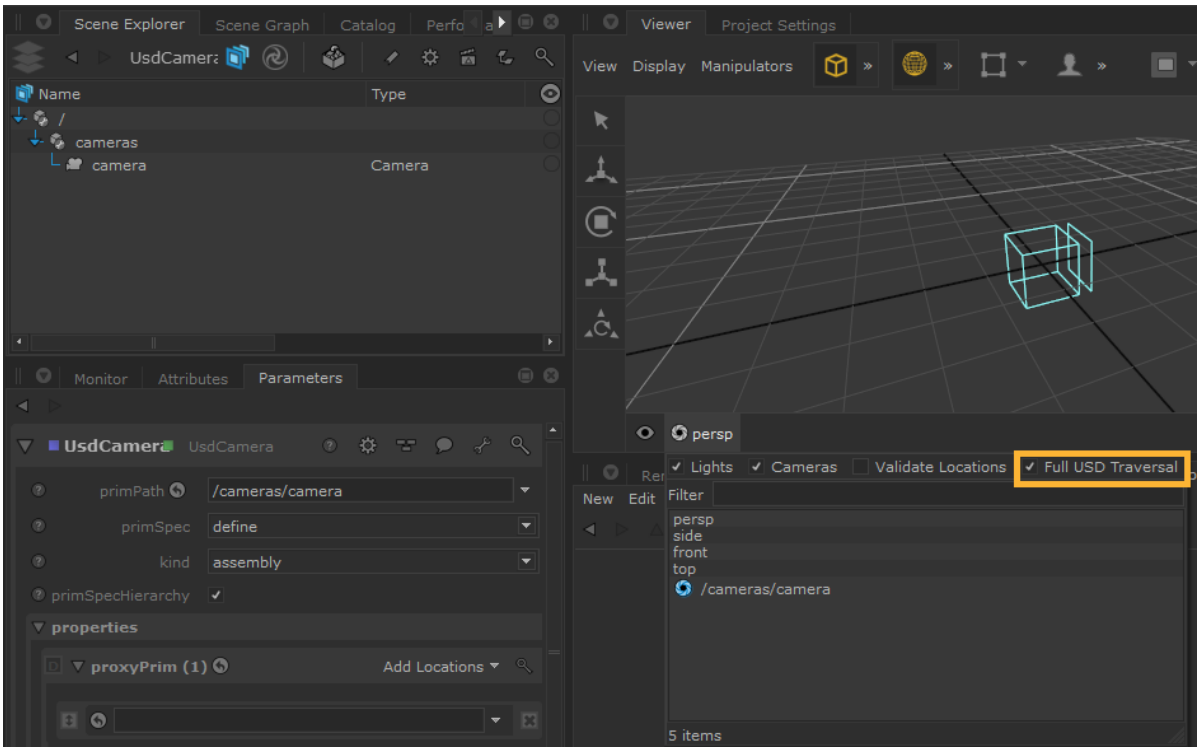
1. Select the **Look Through Cameras and Lights** option at the bottom of the Viewer (this will display the currently viewed through camera, by default it will be set to the perspective camera **persp**).

   This brings up a list of available lights and cameras.

2. Find and select the camera you want to look through. It may help to filter the list to uncheck the Lights checkbox to remove lights from the list.
3. Select the required USD camera from the list.

Cameras not appearing in the dropdown? This Viewer dropdown will be default find USD cameras that are part of the Model Hierarchy. This is for performance reasons, and to save traversing the entire stage, especially when working with large scenes. To ensure cameras are part of the model hierarchy, they (as well as their ancestor prims) require their **kind** to be set to **group**, **component**, or **assembly**. Alternatively, an option is available to enable traversal of the entire stage, removing the need for prims to be specified in the model hierarchy. This checkbox is under the **Look Through Lights and Cameras** menu under the **Full USD Traversal** checkbox option.



Note: Transforming the position of a USD camera while looking through it is not yet supported.

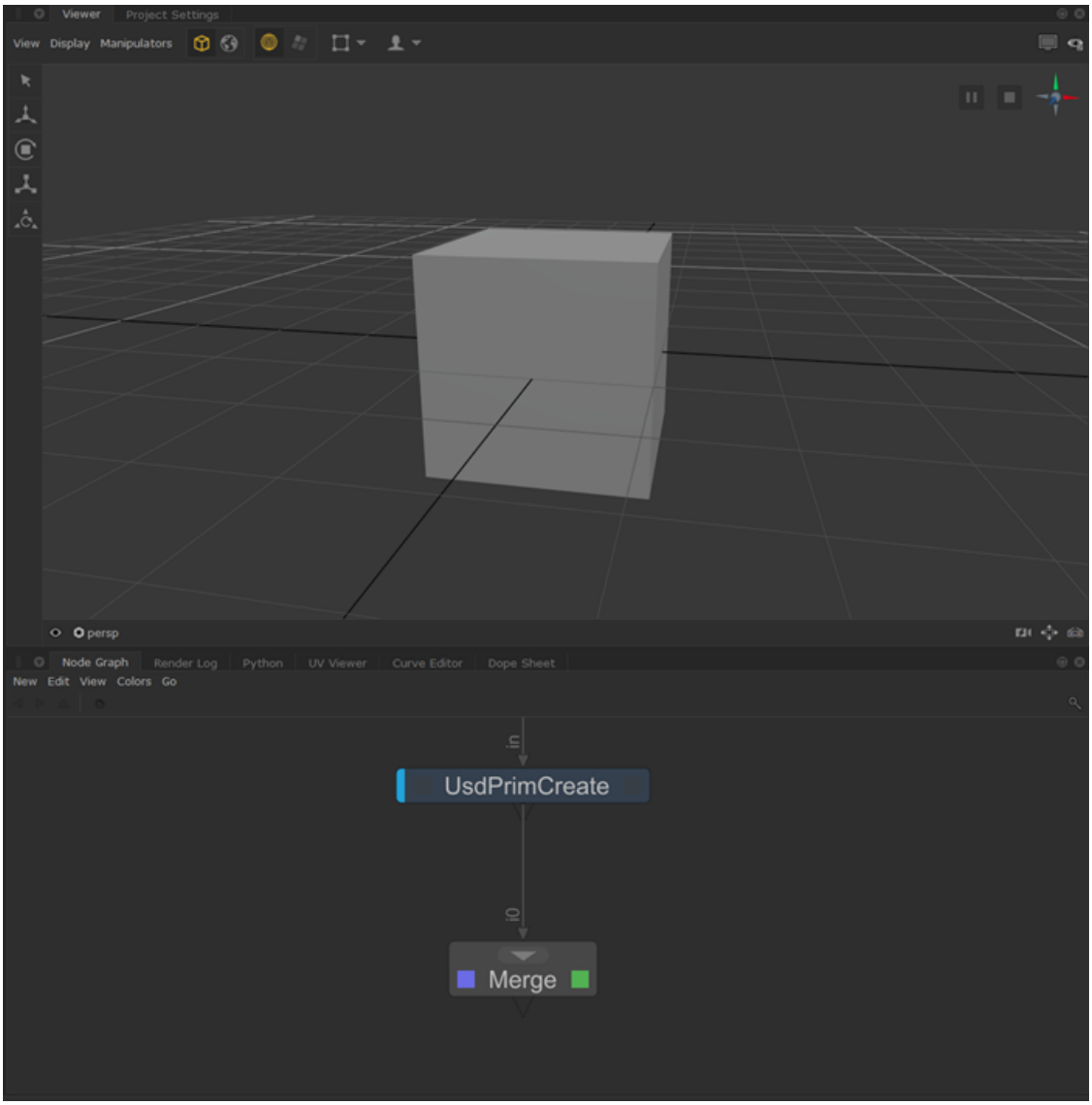
Note: Currently, USD native lights are not yet supported in the **Look Through Camera and Lights** feature.

UsdMaterialAssign Workflows

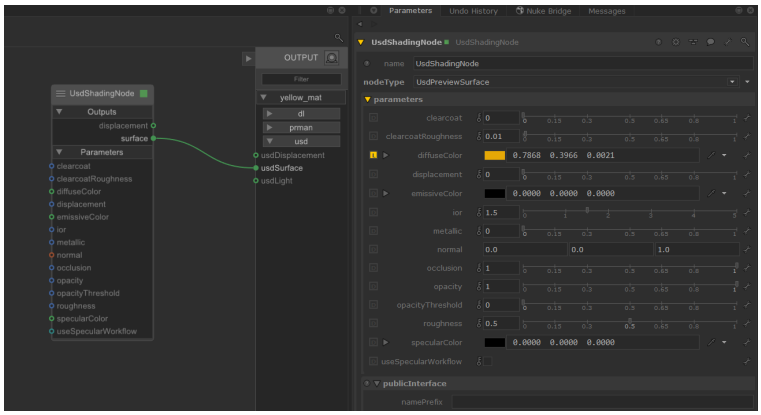
Converting Materials with KatanaToUsd - Practical example

The KatanaToUsd node allows you to create Geolib materials upstream and convert them over USD. With your materials in USD, you can use a UsdMaterialAssign node to assign these to USD Native meshes. Below is a step-by-step example on how this works:

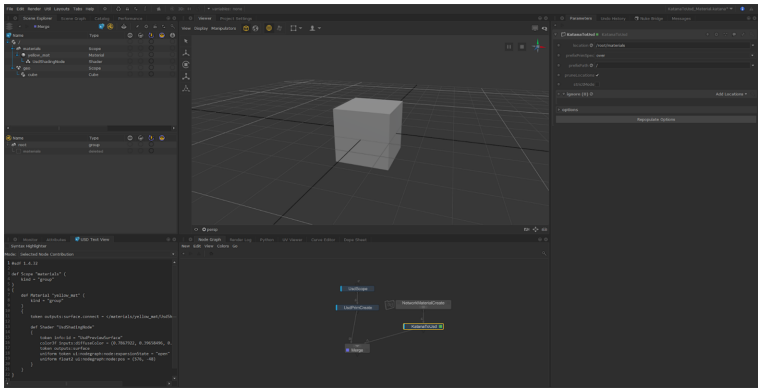
1. Let's start off by bringing in our geometry. Here, we are using a UsdPrimCreate node to create a cube in our scene.



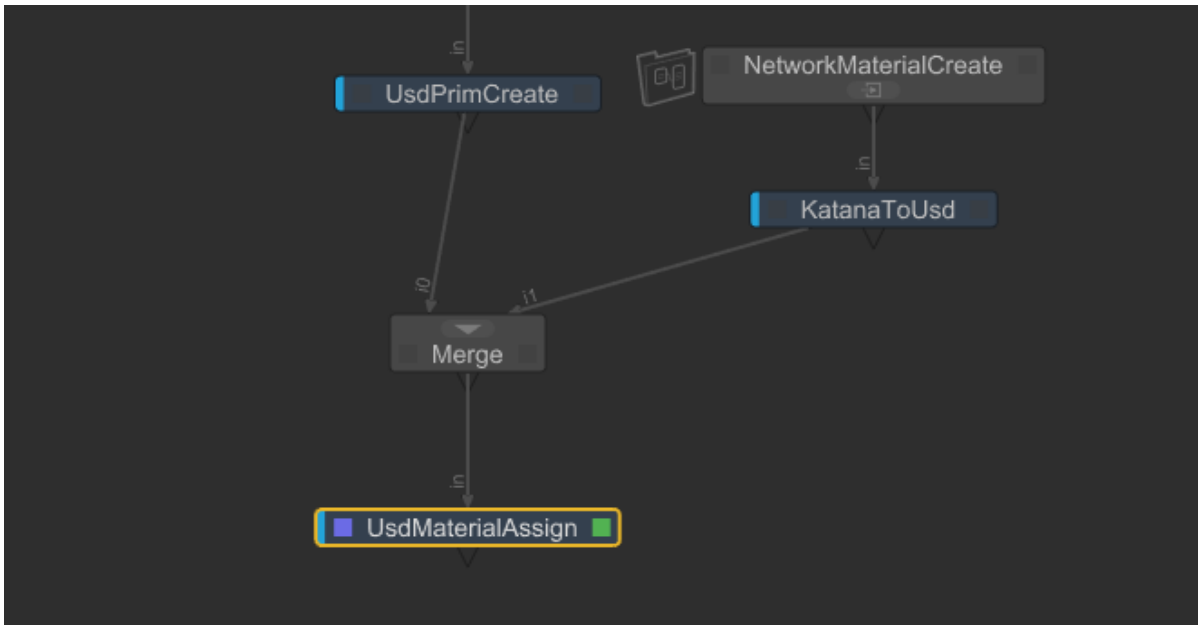
2. Next, we can create a material in Katana using the NetworkMaterialCreate node. Open this node and press **Shift-Tab** to switch to the USD Context. From here, we can hit **Tab**, and type in **UsdPreviewSurface**. Change the color to yellow (or any color you'd like) and connect the surface output to the **usdSurface** input. This should create our yellow material.



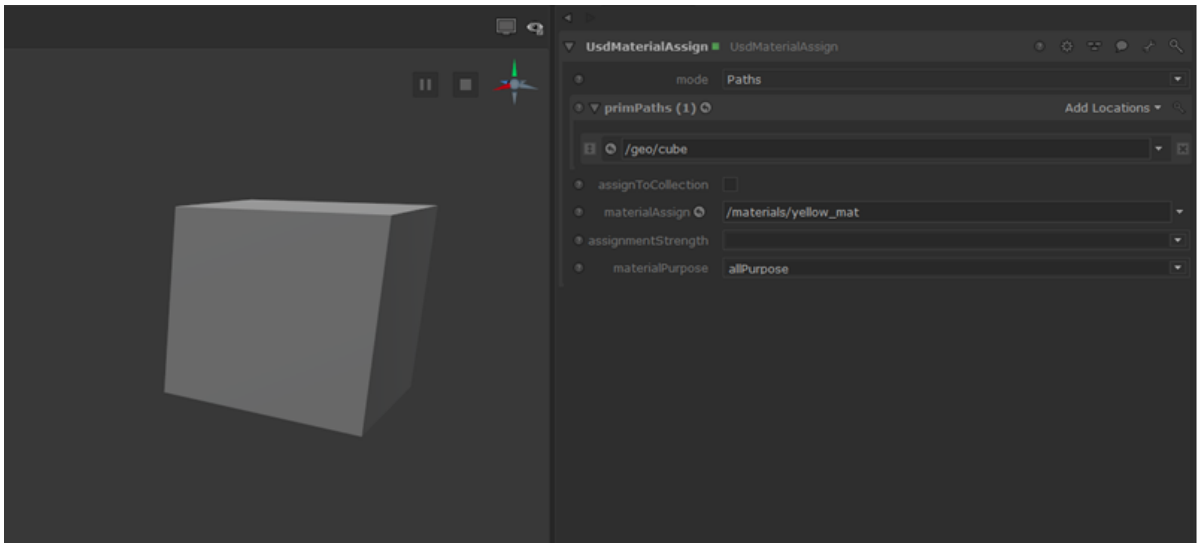
3. Jumping up to node level, we can now add a KatanaToUsd node to convert our Materials from Katana data (Geolib) to USD. Set the location to your material location and connect it to the Merge node. For more information on the KatanaToUsd node, see the reference guide page KatanaToUsd.



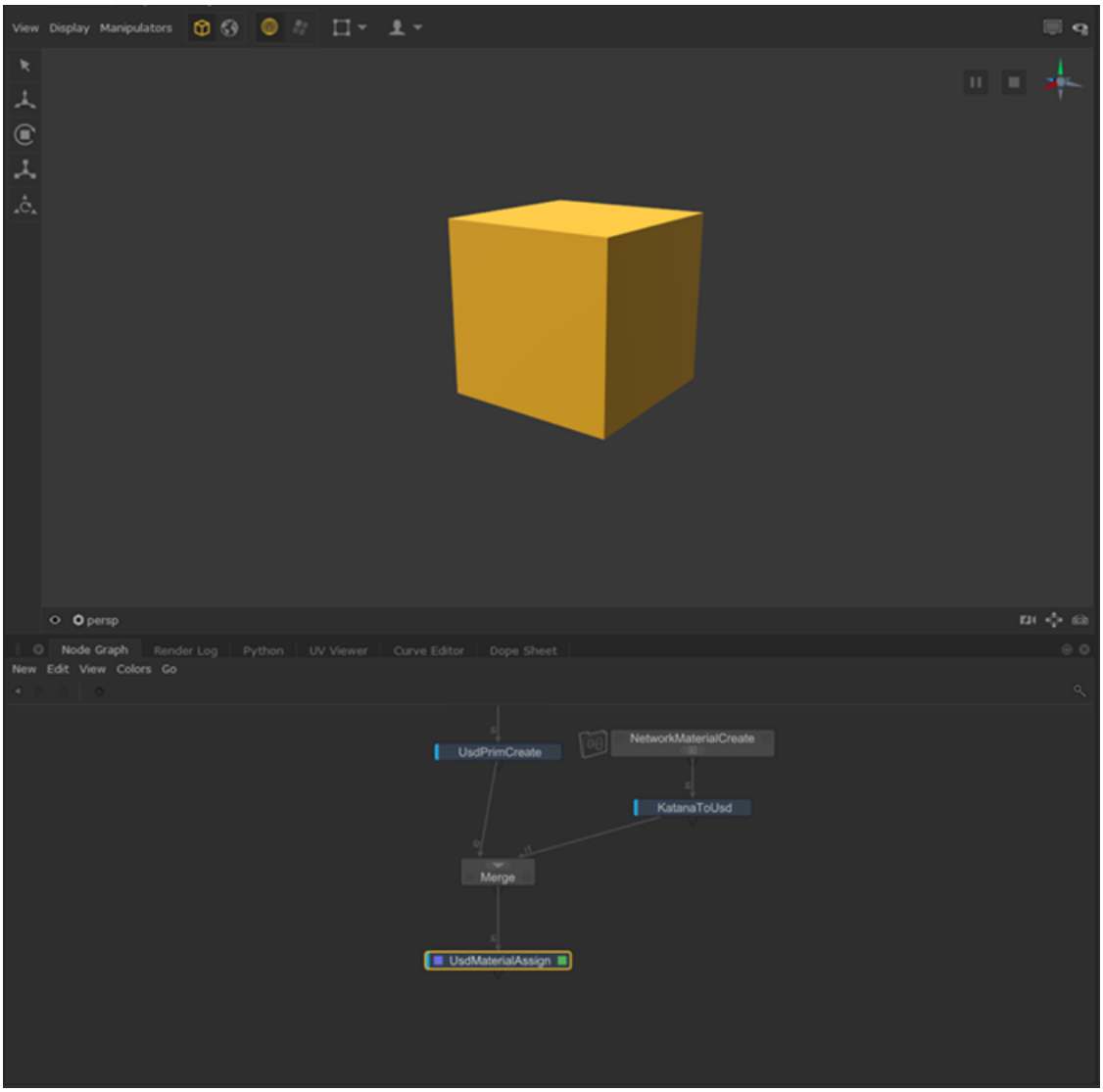
4. With our USD Material ready, we can assign it to our cube. Add a UsdMaterialAssign node and connect it downstream of the Merge node.



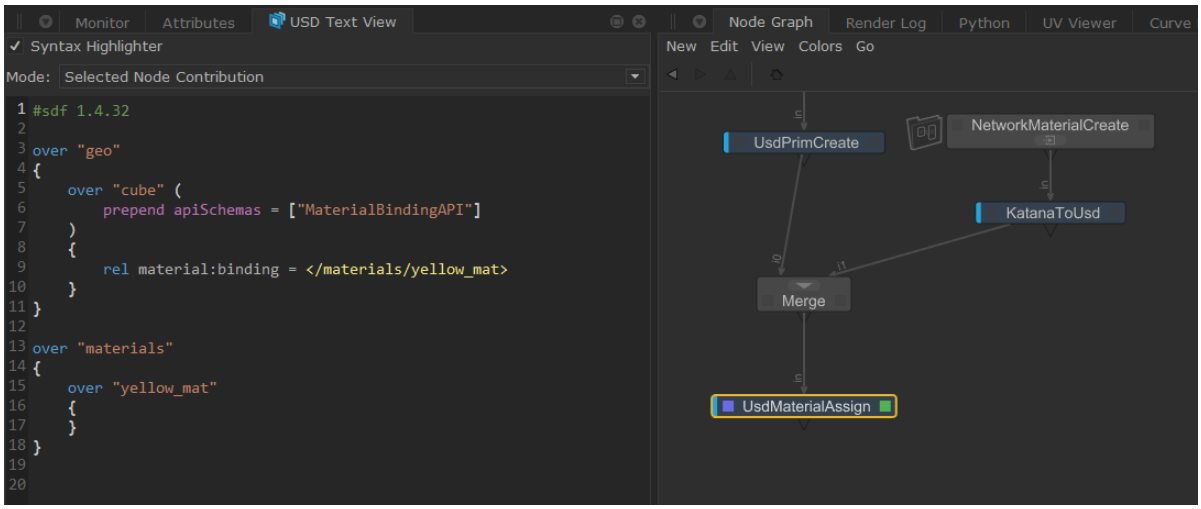
5. Set the **primPath** to the USD mesh primitive you would like to assign your USD material to. In this case, this would be '/geo/cube'. Under **materialAssign**, we specify what material to assign to the mesh. We need to make sure to select a primitive with the material primitive type, which is '/materials/yellow_mat' in our example.



6. Set the view flag to the UsdMaterialAssign node and we should see our cube turn yellow! (If it stays grey, this might be because the **Basic Material** is turned on in the Viewer. To turn this off, in the view tab, go to **Display** > **Basic Material** and uncheck it.)

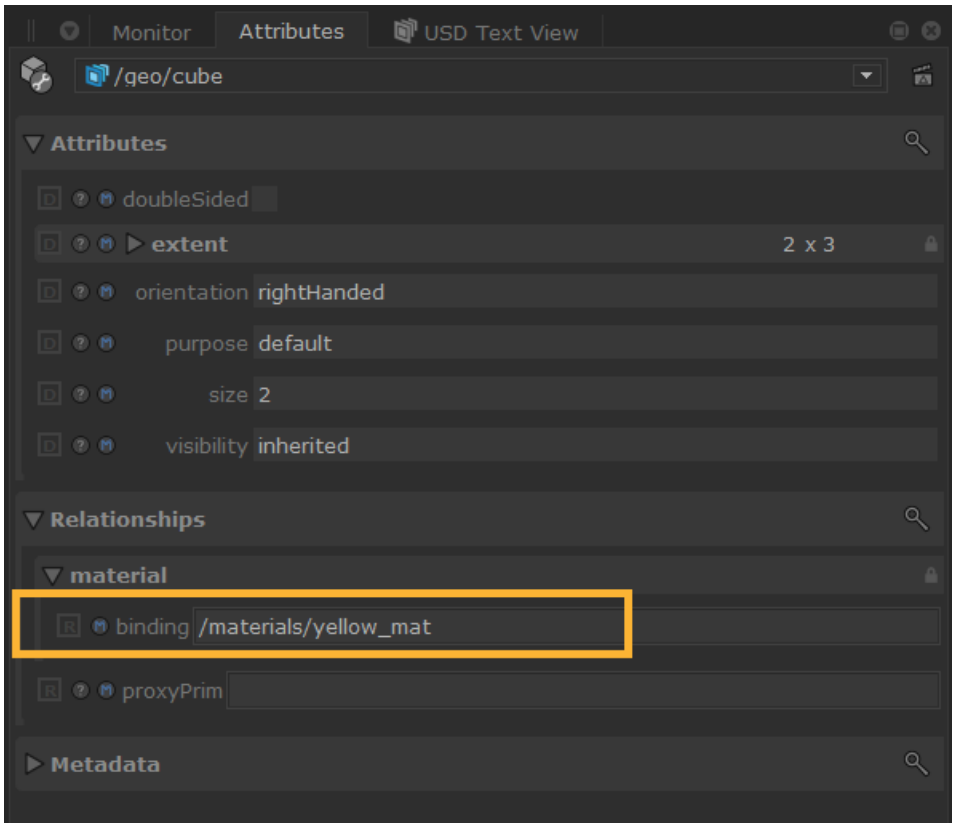


7. To understand more about the material assignment in USD, we can use the USD Text View tab to know exactly what the UsdMaterialAssign node is doing to the stage by selecting the node. The material assignment is stored on the mesh primitive.



The same information can be found in the **Attributes** tab when selecting the cube primitive , under **Relationships** > **material** > **binding**.

See Attributes tab



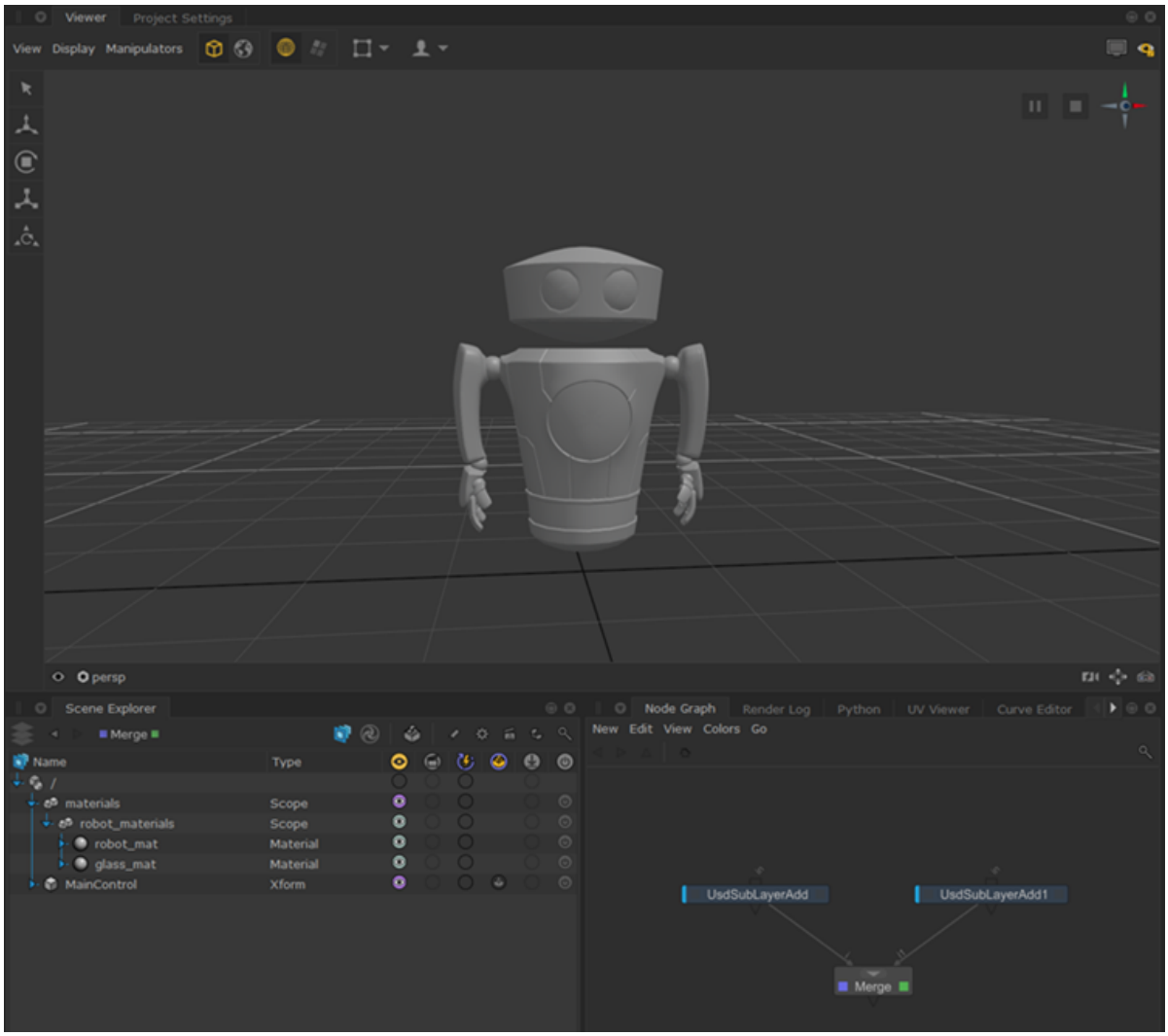
Incoming USD Files - Practical example

In our previous example, we showed how to create materials in Katana, convert them to USD and assign them to USD meshes. Next, we'll have a look at how this would work if you have saved out your scene in separate USD layers, e.g. a layer for the mesh, a layer for the materials, a layer for lights, etc. The assets we'll be using can be found in the example projects in Katana under **Help** > **Example Projects** > **Native USD** > **Native USD Look Dev Example**.

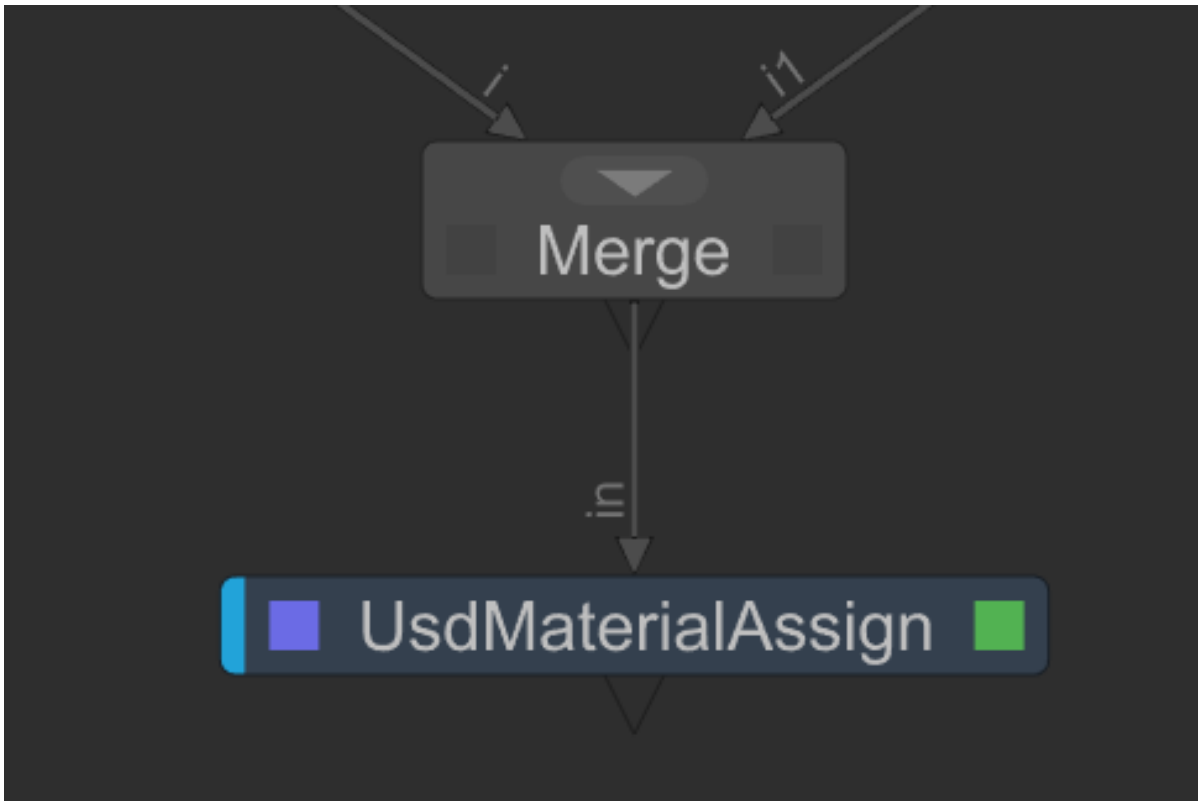
Here's a quick look at the .usda files we are using. Our first file, robot_asset.usda, consists of the robot mesh (which is sub layered in for clarity reasons) and a collection. The second file is solely made up of materials.



1. Let's use the UsdSubLayerAdd node in Katana to get both these files in our scene. Merge both UsdSubLayerAdd nodes together by selecting them both and hitting "M".

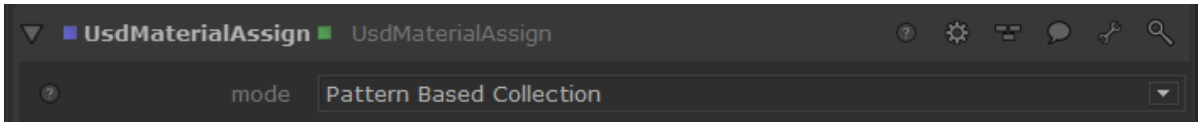


2. With our scene populated, we can assign our materials to the robot. First, let's assign our robot_materials. Start by adding a UsdMaterialAssign and connecting it downstream of our Merge node.

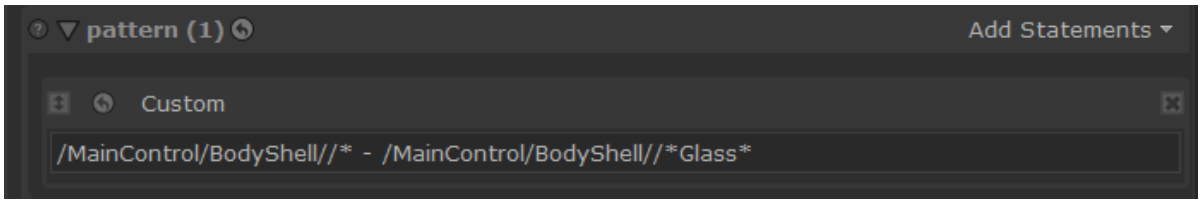


Assigning Materials using PBC

3. Highlight the node and specify the mesh you would like your material to be assigned to in the **primPaths** widget. You can assign the materials to prims, as we saw in the example above.
   For the purpose of this exercise, we will use Pattern Based Collections to see how you can assign materials based on a set of rules. We want our robot material everywhere, except on the primitive that are made out of glass, which is where our glass material will go. We can do this by using a custom expression to include all primitives, unless it has 'glass' in its name. Switch the **mode** on the UsdMaterialAssign node to **Pattern Based Collection**.

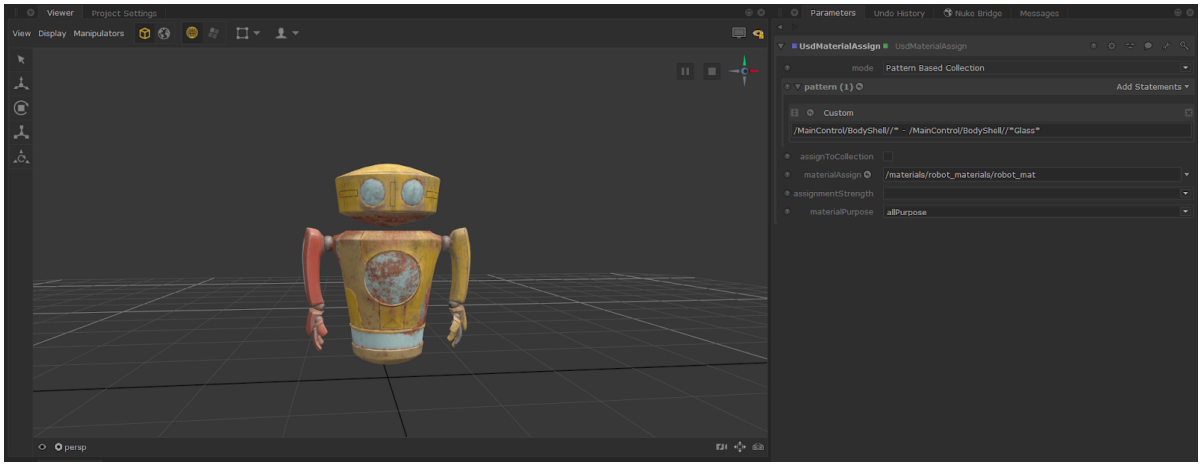


4. Under **Add Statements**, we can select **Custom** which will allow us to create a custom expression. Insert "/MainControl/BodyShell//* - /MainControl/BodyShell//*Glass*" which translates to "include everything under ../BodyShell, except primitives with the word 'Glass' in its name".



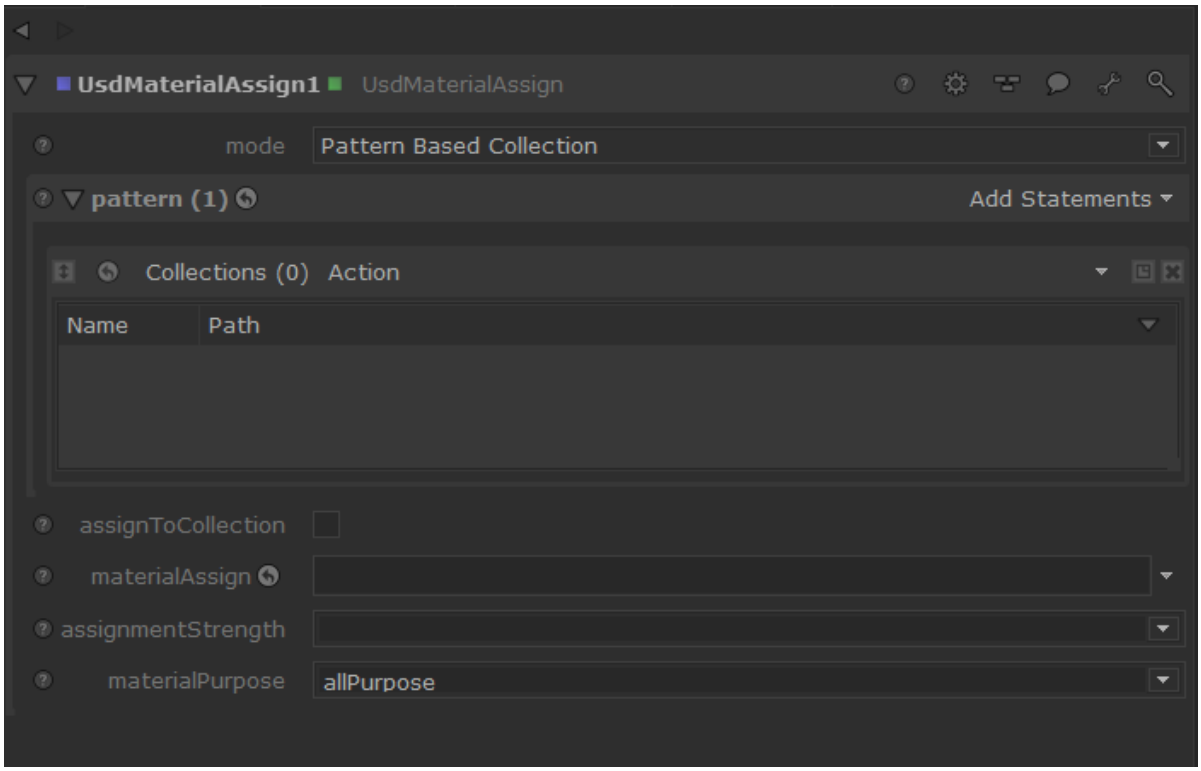
To learn more about statements you can add with PBC, see Pattern-Based Collections with USD.

5. Now we can assign our material. We'll drag our **robot_mat** material into the **materialAssign** parameter. (If we don't see it update immediately, that means a **Basic Material** is being applied in the viewer. Turn this off by unchecking **Display** > **Basic Material** in the Viewer.)

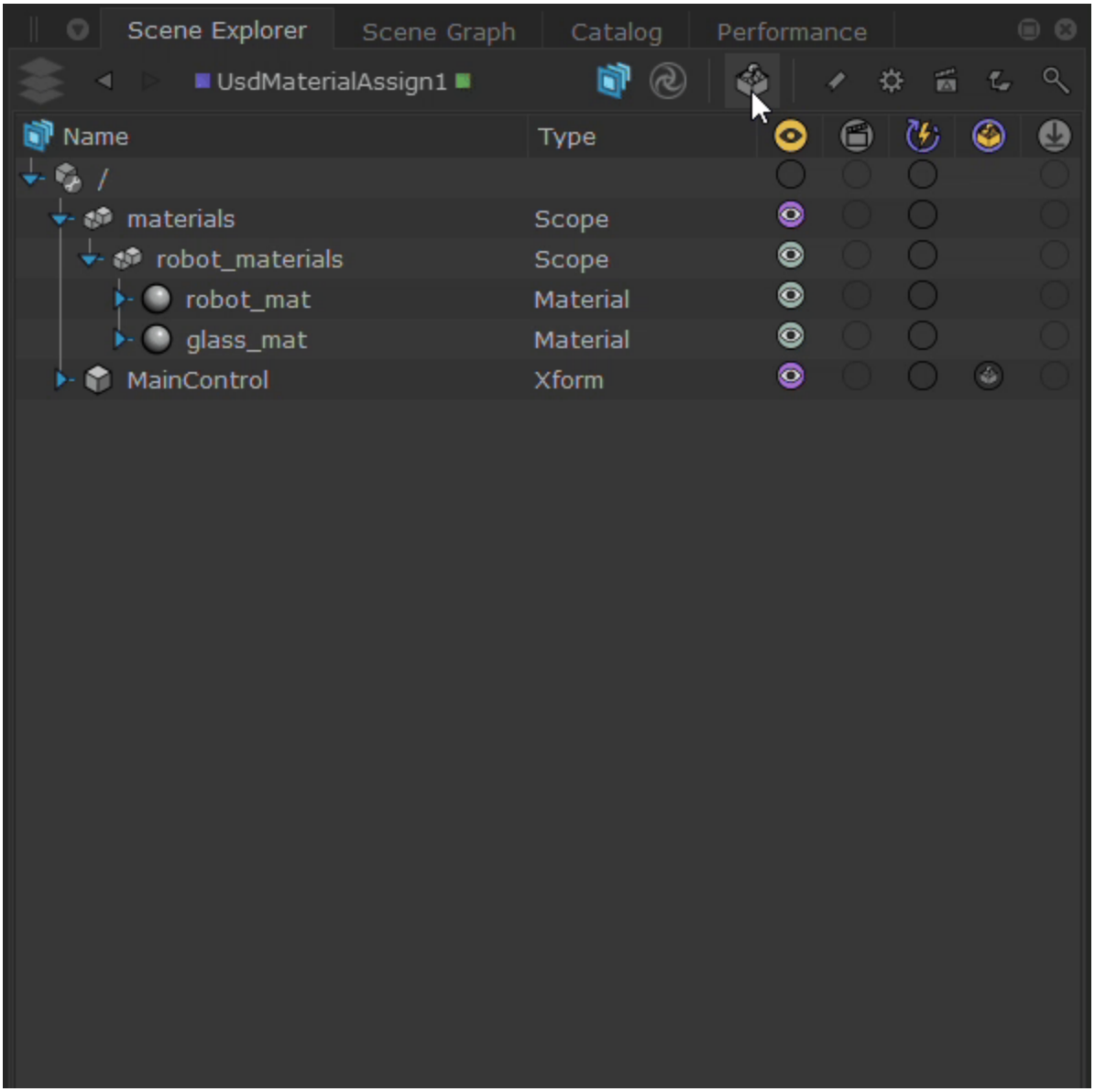


Materials and Collections

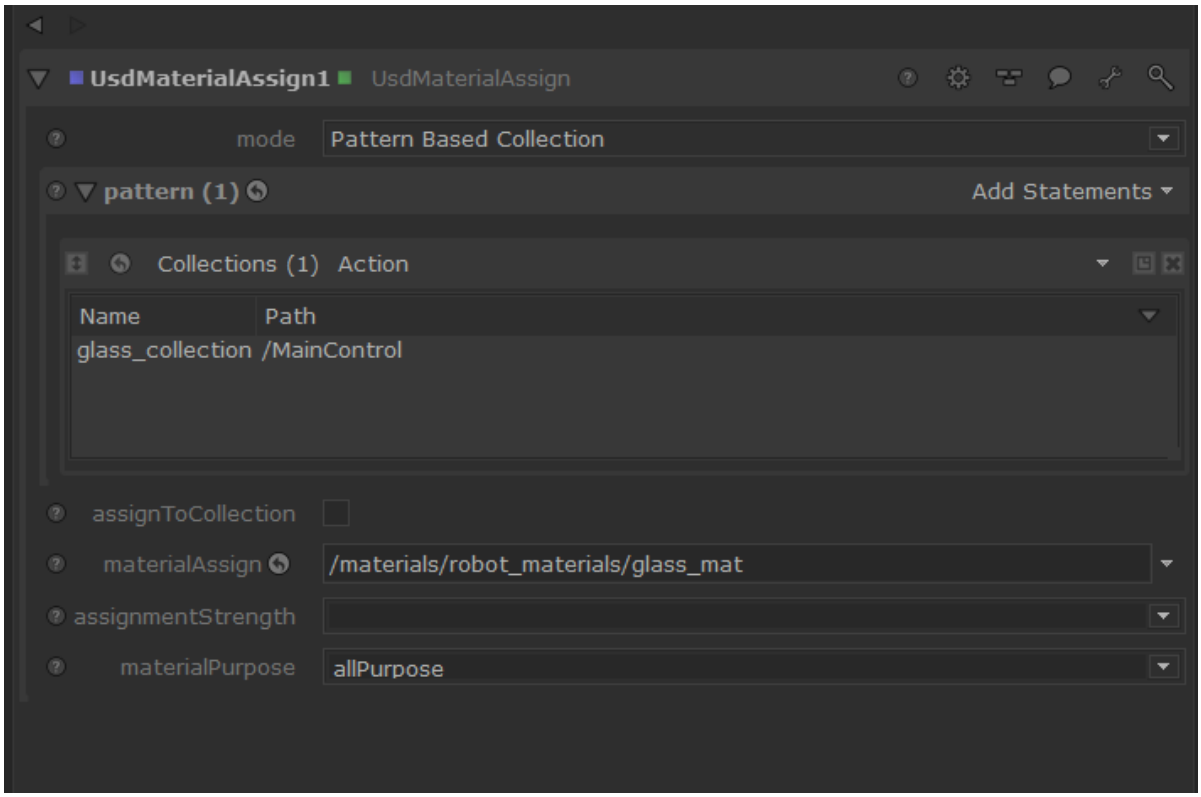
6. As you can see, we have some missing sections of the lookdev on our Robot. In this step, we're going to show another way you can make assignments, using **Collections**. We'll use it to assign our glass material to specific parts of the mesh. Add a UsdMaterialAssign node downstream and connect it. Change its **mode** to **Pattern Based Collection** and **Add Statement**.



7. Open the Collections panel in the **Scene Explorer** by clicking the Collections Icon. We can choose what Collections to show by enabling the **Collections working set** on the primitive it is on.



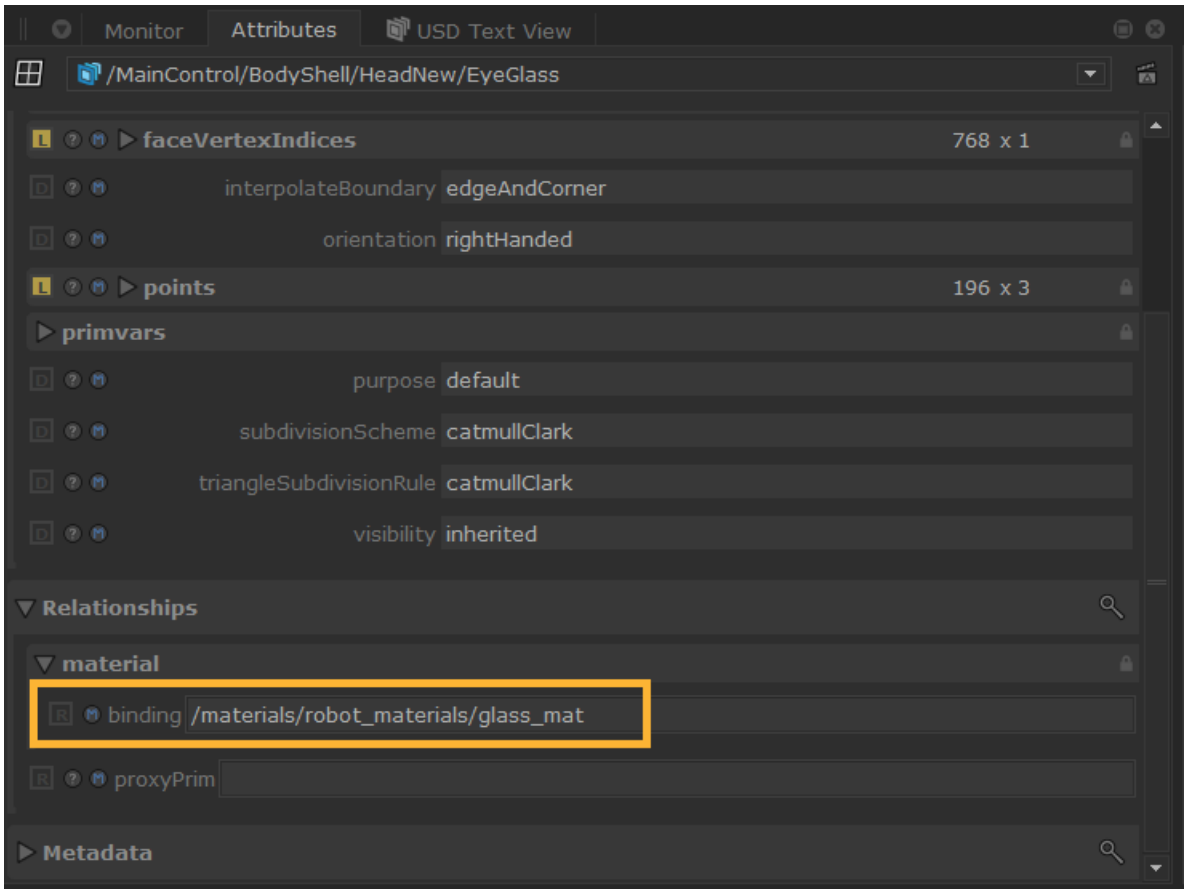
8. From here, we can drag-and-drop our **glass_collection** into the **pattern** widget and assign our glass material.



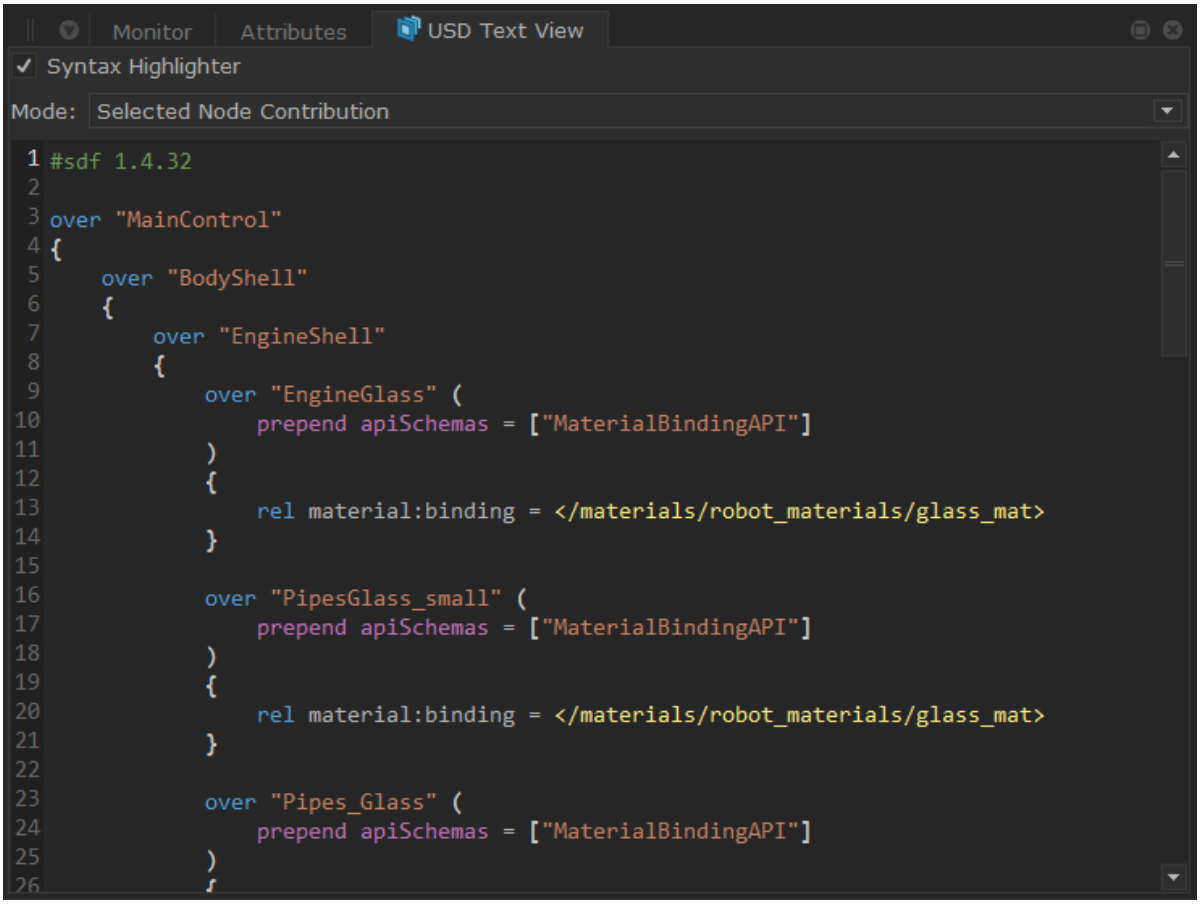
9. We can now see both the robot material and the glass material assigned to our robot!



10. If we take a look at the **Attributes** tab with a mesh primitive selected, we can see that our material binding is correctly being created on the mesh. To get a better understanding of what is happening under the hood, we can open the USD Text View tab and select one of the UsdMaterialAssign nodes.



See USD Text View tab



Using the Scene Explorer

We introduced the Scene Explorer tab as part of our growing support for USD, making navigating and exploring scenes easier. The scene explorer contains panels for two scene graphs:

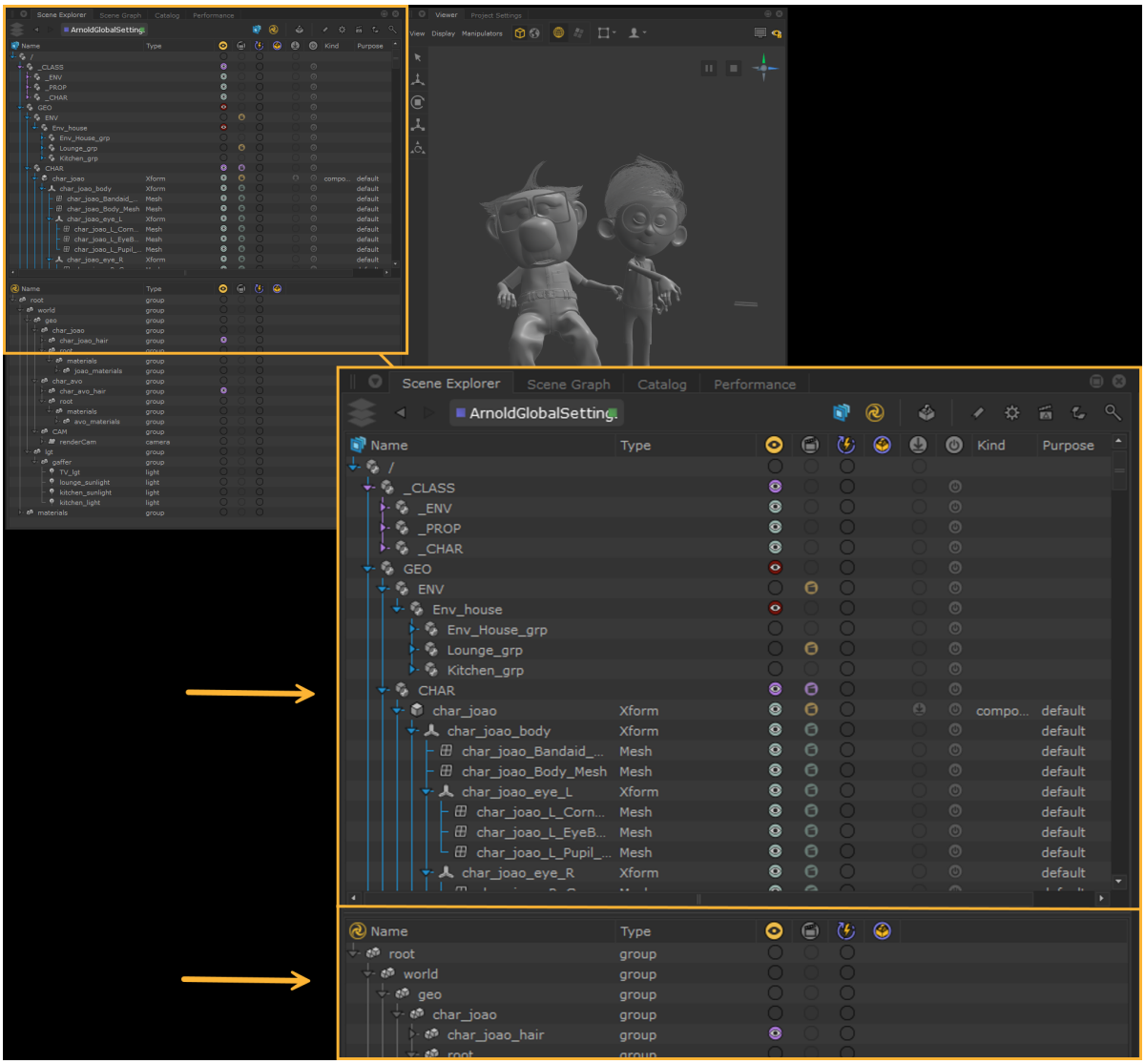
- **USD Scene Graph**- Inspect and manipulate the properties and attributes of prims in the stage hierarchy.
- **Katana Scene Graph** - The Katana Scene Graph for non-USD items.

There is also a panel above the scene graphs, showing the **Collections** in the project. This is collapsed by default. See more about this at Collections in the Scene Explorer.

Note: The Katana Scene Graph is explained in Using the Scene Graph.

The workflow for the Scene Explorer will be familiar if you've used the Katana Scene Graph. Both the USD Scene Graph and the Katana Scene Graph panels in the Scene Explorer share working set columns for essential operations, including visibility in the viewer and renderer. However, the USD Scene Graph adds two USD-specific working set columns so that you can have more control over payloads and prim activation.

As you can see in the image below, the **Scene Explorer** tab includes both the USD Scene Graph and the Katana Scene Graph beneath.



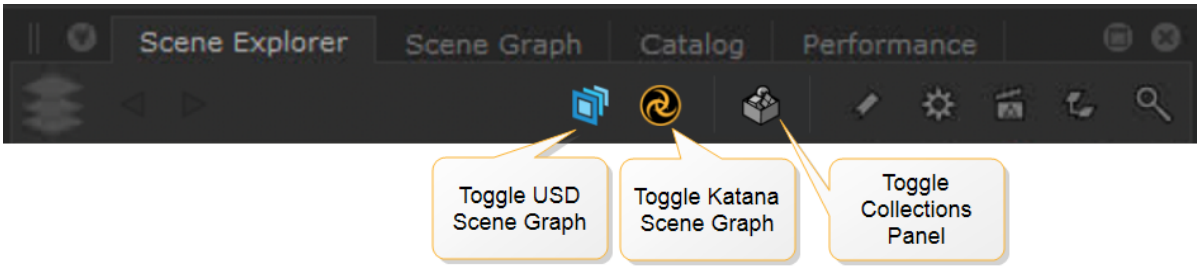
The Scene Explorer. The top section is the USD Scene Graph, and the bottom is the Katana Scene Graph.

Each scene graph is identified by the USD or Katana logo in the top-left corner of the panel next to the **Name** column.

Note: To hide the Katana Scene Graph in this panel, either drag the top of the panel bar either to the top or bottom or by toggling either the USD or Katana logo located at the top of the panel.

Viewing Scene Graphs

Use the three selection buttons at the top of the Scene Explorer tab to display or hide the USD Scene Graph, or/and the Katana Scene Graph, or/and the Collections panel.

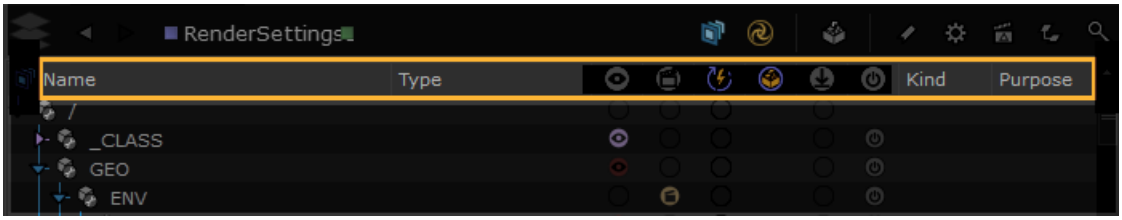


If a panel is hidden, its icon is greyed-out.

USD Scene Graph Columns

Data Columns

The USD Scene Graph part of the Scene Explorer tab has four columns of key information for each item:



- **Name** - The name of the prim.
- **Type** - The data category of an element.
- **Kind** - The organizational role of an element in the hierarchy. It is categorized as values such as **component** for leaf level items or **assembly** for a collection of components. This can be modified using the UsdKindSet node in Katana.
  See Kind in Pixar's USD Glossary.

- **Purpose** - The category for visibility of components or primitives. For example, a **default** purpose generally means that it is a primary element of the scene, and is involved in various processes, such as rendering, simulation, and interaction within the viewport. A **proxy** purpose would indicate its use in viewports but not in renders, whereas a **render** purpose indicates use in renders but not viewports. See Purpose in OpenUSD's USD Glossary.

By default all purpose types are displayed in the Viewer. However, you can choose which are shown by selecting **Display > Purposes** in the **Viewer** tab.



Note: When prims with the proxy purpose are added to Katana, by default they have their render visibility turned off.

Common Working Set Columns

The USD Scene Graph shares four working set columns with the Katana Scene Graph:

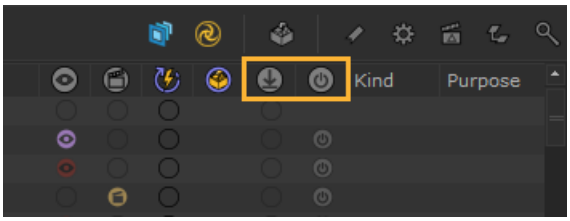


The Viewer Visibility, Render Inclusion, and Live Render Updates columns control visibility in the viewer and renderer. To find out how to work with these, see Working Sets.

The Collections working set shows which locations have a collection on them. See Collections in the Scene Explorer.

USD-Specific Working Set Columns

The USD Scene Graph includes two columns for payload management and prim activation/deactivation. See Payloads in the Scene Explorer, and Activating and Deactivating Prims.



Styling in the Scene Explorer

Prim Specifiers determine the order of operations when building up a USD scene. Seeing whether prims are **defined**, **over** or **class** helps you to quickly debug your USD composition.

The Scene Explorer allows you to quickly gain visibility of your scene setups, via visual cues to indicate information such as prim specifiers, visibility, instance proxies..

In the Scene Explorer, the tree lines **styling** will change based on certain prim rules:

- **Defined** prims show a **blue branch** in regular font
- **Abstract (class)** prims show a **purple branch**
- **Instance proxy** prims are displayed as **dotted lines**

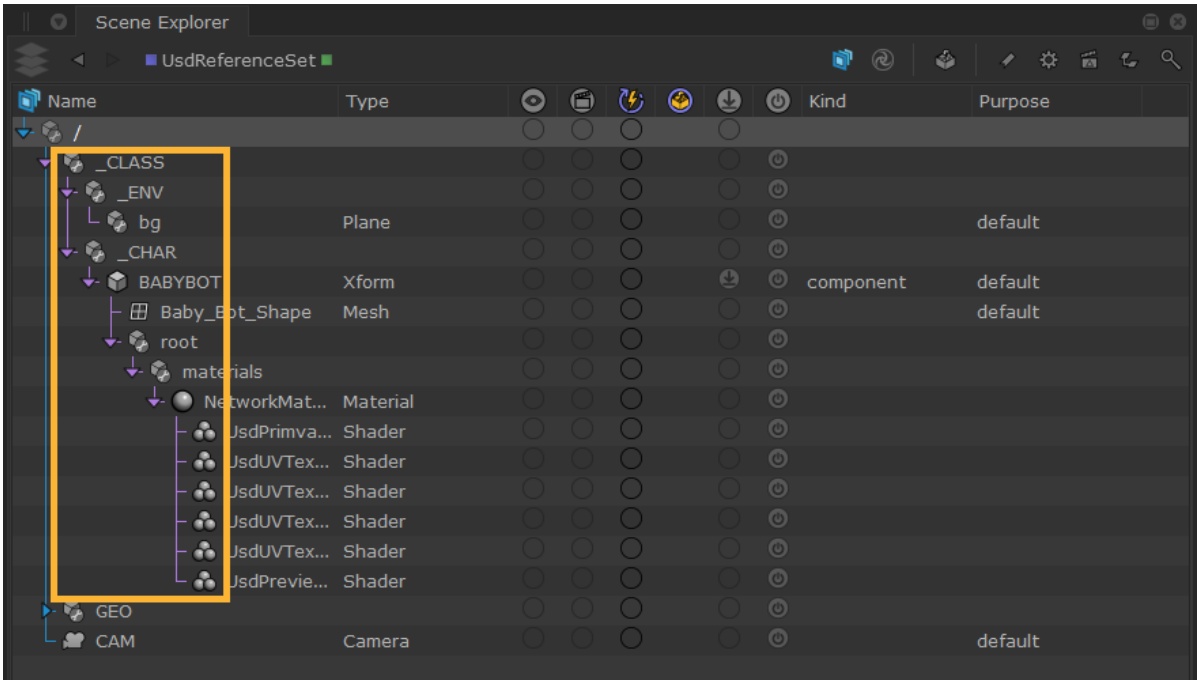
Additionally, font styling is used to indicate the following:

- **Over** prims in undefined locations show a **dimmed** font
- **Inactive** prims show a **dark strike-through** font

Abstract prim - Example:

Abstract prims can be locations under a **class**, such as the example seen below, and are often utilised to tentatively bring assets in the scene, to be later placed on the stage. A **class** prim defines a prim meant to act as a template for other prims. They ensure uniformity and reusability throughout the scene by allowing other prims to reference, inherit or specialise it, to inherit its characteristics.

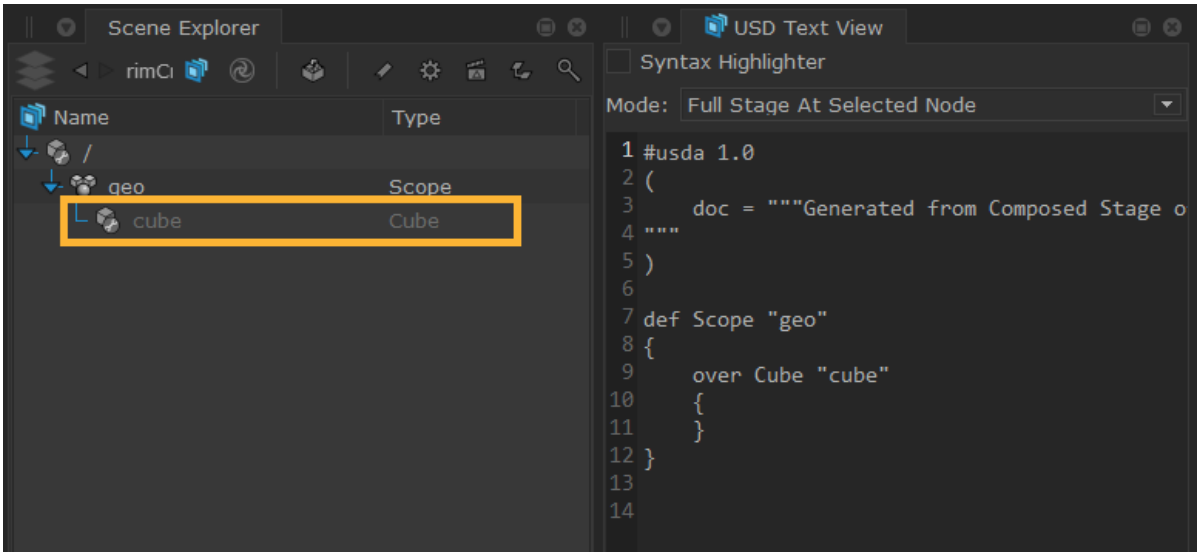
Abstract prims will be present in the usd document, however will not be present on the stage. Therefore abstract prims indicated via **purple** hierarchical lines in the **Scene Explorer**, cannot be previewed in the **Viewer** tab.



Note: This image showcases an example that can be found under **Help** > **Example Projects** > **Native USD** > **Importing and Building USD stages.**

Undefined over - Example

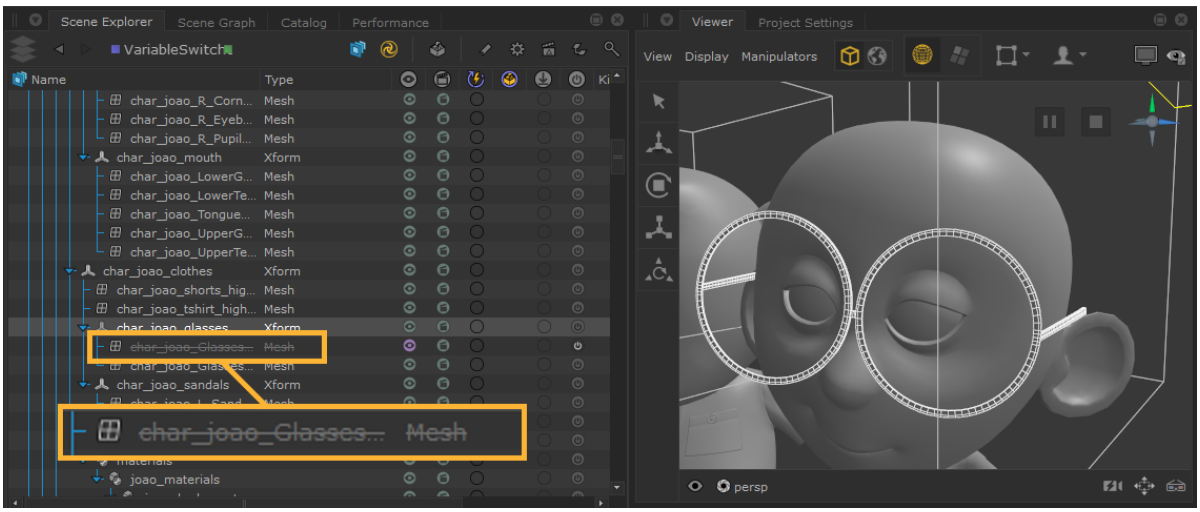
An **undefined over** is an override made on a primitive that is not defined. Here we are doing an override on a cube, but this cube was not previously created with the "def" **primSpec**. These **undefined overs** are shown in the **Scene Explorer** as **dimmed**.



Inactive locations - Example

Inactive locations will be seen in the Scene Explorer as **dimmed** and **strike-through**.

In the example, the lenses of the glasses have been set to **inactive** in the incoming USD document. This is indicated via the Scene Explorer, the same way that pruned locations are seen in Katana's geolib's panel in the Scene explorer.

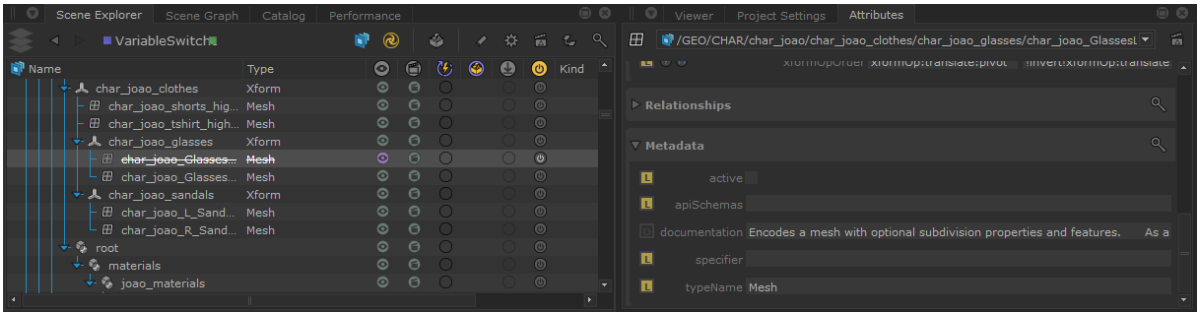


Note: Locations in USD cannot be deleted! The closest thing it has to removing elements permanently is by setting the active property on the prim location.

Another way you can identify inactive prims, is through the **Active working set** column . The icon against the **Active** column will have changed if a prim is inactive:  See Activating and Deactivating Prims.



Another way to see whether a prim is active, is through the **Attributes** tab, under the **Metadata** section. See Inspecting USD Properties Using the Attributes Tab.

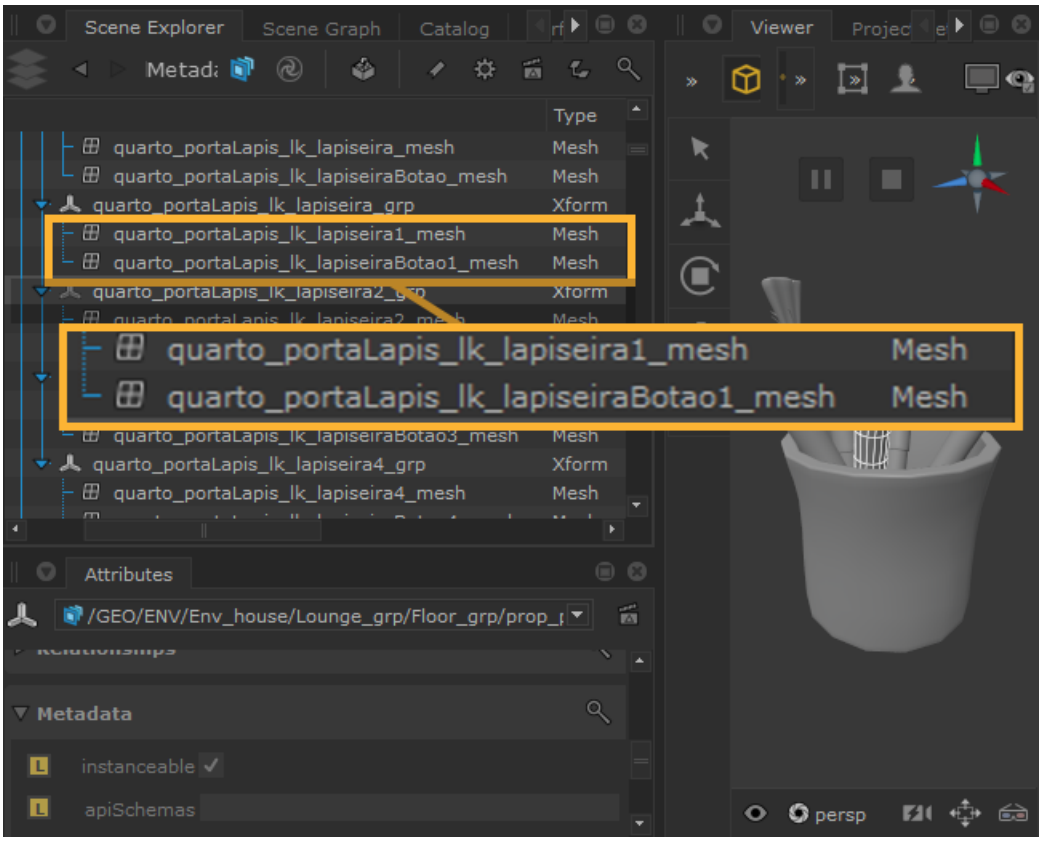


Tip: The UsdActiveSet node can be used to make a location either inactive or active, this will persist in the scene and output USD document. Alternatively, the actions made in the Active working sets within the Scene Explorer are temporary.

Instance proxies - Example:

Additionally, prims that have the property **instanceable** set to true, will be indicated in the Scene Explorer via **dotted lines**. This is to indicate prims that are an instance (an uneditable copy) of another prim location. This means any changes made to the instance proxy prims (prims that are downstream) will be ignored.

In this example, the pen and the pencil pot are all derived from the same prim, and therefore are all instances of that prim. This is a great way to build your stage in USD for prims that are repeated many times, as it can greatly improve performance.



To learn more about Scene Graph Instancing, refer to Scenegraph Instancing in OpenUSD's USD Glossary.

To learn more about Prim Specifiers, see Specifier in OpenUSD's USD Glossary.

Payloads in the Scene Explorer

Payloads for Selective Stage Loading

Katana makes use of **deferred loading** to improve performance. When working with large datasets, portions of the data can be kept in reserve until needed, thus freeing up computational resources for more pressing tasks. In the Katana Scene Graph this is achieved by selectively expanding branches of the tree. In the USD Scene Explorer, it is achieved using **payloads**.

Loading and unloading payloads triggers the recomposition of the USD stage. When a payload is loaded, it brings in new data, such as prims and properties to the stage. Similarly, when you remove a payload from a stage, recomposition is triggered as data is removed from the scene graph.

A **payload** is a reference to scene information that can be loaded into the stage when needed. Any prim can contain one or more payloads, or have none.

For example, imagine you have a tree asset that is highly detailed. You want to include multiple instances of the tree in a forest.

Your tree asset looks like this:

```
#usda 1.0
def Xform "Tree" {
    def Mesh "Trunk" { /*... mesh details ...*/ }
    def Mesh "Leaves" { /*... mesh details ...*/ }
    // ... Perhaps more variations or details like fruits, flowers, etc.
}
```

To include trees as payloads in your forest, the USD file could be written as follows:

```
#usda 1.0
    ()

def Xform "Forest"() {

    def Cube "Tree1"
        (

            prepend payload = @. / tree.usda @ < /Tree>
        ) {
            double3 xformOp: translate = (0, 0, 0) uniform token[] xformOpOrder =
["xformOp:translate"]
        }

    def Cone "Tree2"(
        prepend payload = @. / tree.usda @ < /Tree>
    ) {
        double3 xformOp: translate = (5, 0, 10) // Positioned for natural tree spacinguniform token
[] xformOpOrder = ["xformOp:translate"]
    }
}
```

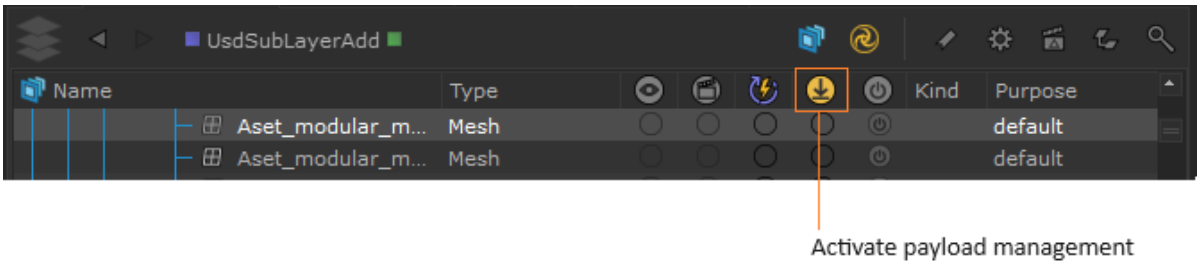
Using payload enables selective loading and unloading of specific portions of a USD stage to speed up your workflow. Loading any prims or layers that are not currently needed can be deferred until required.

Example: Consider working with a large cityscape. Instead of loading all the buildings, cars, and trees simultaneously, which can be resource-intensive and slow, you can use payloads to load only the parts of the city that are currently visible or necessary for the scene. As you shift the focus of your work, you can revise the payload loading options to select the parts you need.

Deferred loading should be familiar to experienced users who have worked with the Katana Scene Graph. As part of the USD workflow, we have incorporated deferred loading using payloads into the Scene Explorer.

Loading and Unloading USD Payloads

Payloads are activated by selecting the Payload button in the Scene Explorer toolbar.

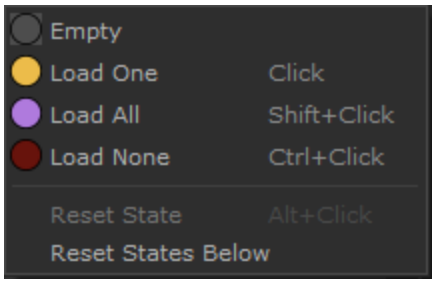


When this payload button is deactivated, payloads are loaded via expansion of USD scene graph locations, mimicking Katana's deferred loading workflow. When expanding prim locations using the deferred loading method, payloads will load if they are present on that prim, but will not load any payloads on its decendants.

Prims with payloads have a down arrow icon in the Payload column.

Payloads are managed by assigning a rule to a prim. This rule determines if payloads are to be loaded for the prim and its descendants.

There are three stage load rules available, as shown in the image below. Either right-click the dot for the options or use the mouse-click combinations listed.



Right-click in the payload column to open the context menu.

Note: Though named differently, the three stage load rules map directly to the stage load rules found in USD. Load One is the OnlyRule, Load None is the NoneRule, and Load All is the AllRule.

The rules only apply if there are payloads on the stage. When there are no payload the rules have no effect.

Prim Payload Status

The color of both the arrow and the background change to indicate the payload status on a particular prim. The table below shows the possible states.

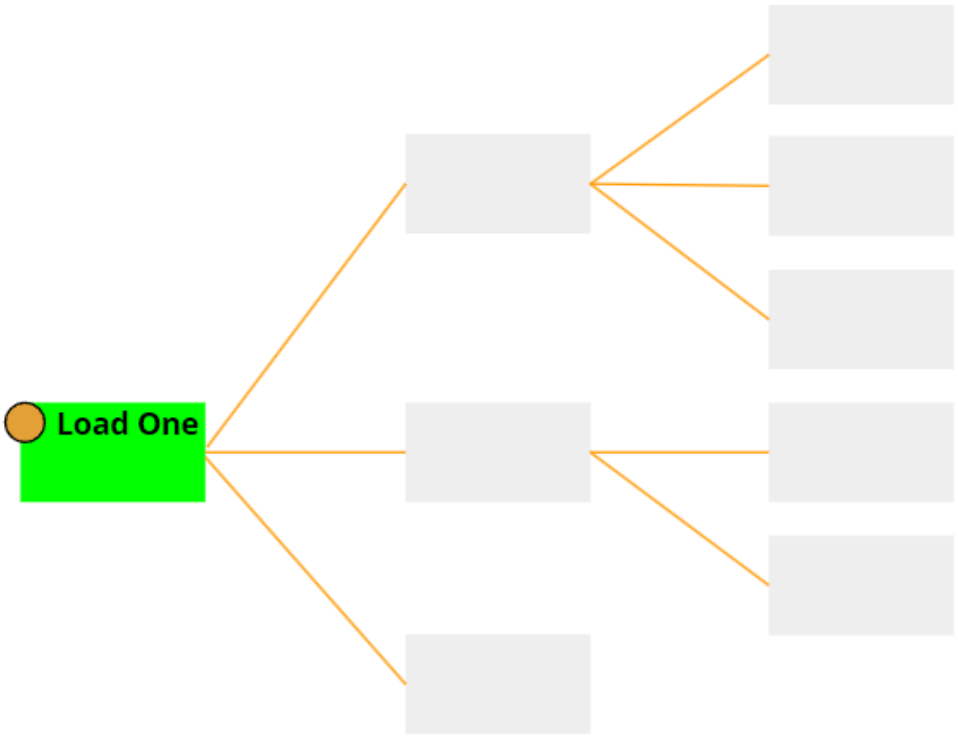
Payload Rule	Payload Management Column Active	
	Loaded	Unloaded
Empty		
Load One		
Load All		
Load None		
By Inheritance (loaded due to implicit loading by ancestors or descendant load rules)		n/a

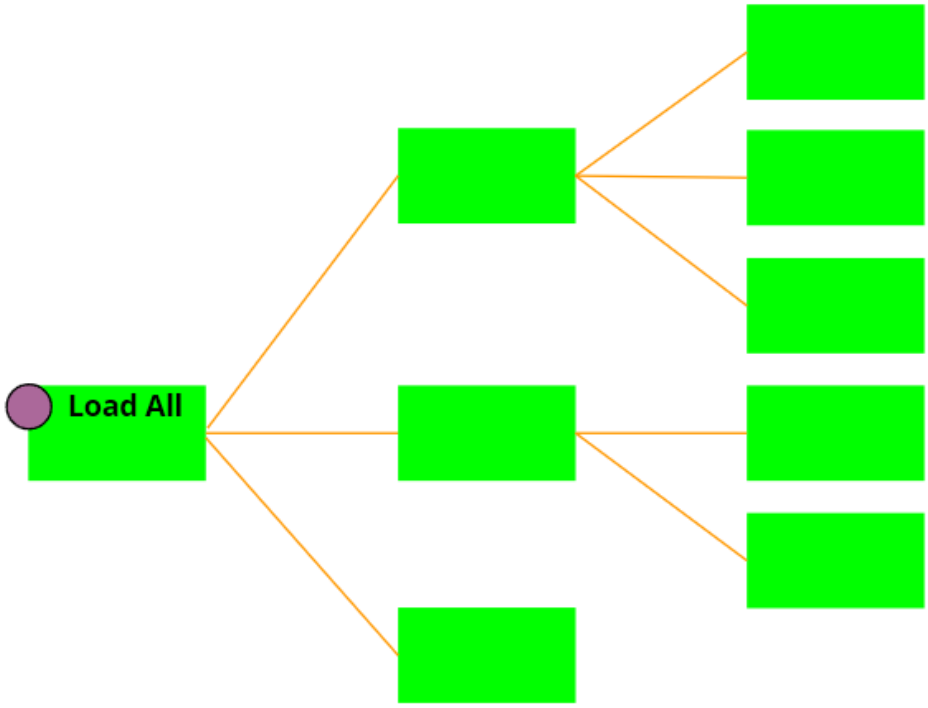
If the prim has been deactivated, Katana desaturates the icon and draws it in a grey circle.

Effects of Stage Load Rules

The table below describes how payloads are managed according to the stage load rule on the prim.

To help understand how the rules affect payload loading, the tree diagrams in the table below represent some simple scenes. Each box is a prim and the connecting lines show the prim hierarchy. A green box indicates if there is a payload at this prim, then this payload is loaded. A grey box indicates if there is a payload at this prim, then the payload is not loaded.

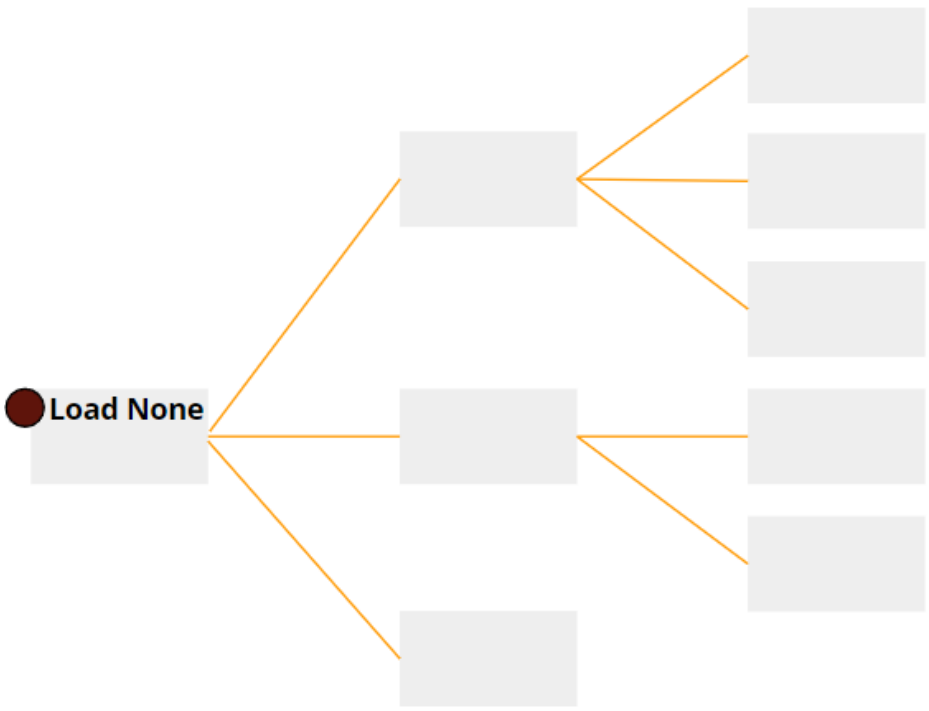
Load One Click to toggle	Load payloads on the prim but not its descendants  Only the payload on the prim in green is loaded. Example: Suppose the scene is In an old church, full of details like candlesticks, icons, stained glass windows, and furniture. You choose to work initially on the daylight shining through the windows. By using the Load One payload method you load only the structural components and windows, reducing memory overhead and enhancing viewport performance. This allows you to fine-tune the primary light dynamics without the computational overhead of secondary assets like books and artifacts.
Load All Shift + click	Include payloads on the prim plus all its descendants



Payloads on the prim and on all descendant prims are loaded if present.

Example: In the church example above, once you have addressed primary lighting behaviour, you could then use Load All to bring in all other objects and address more detailed lighting challenges.

Load None **Ctrl** + click	Exclude payloads on the prim and all its descendants.



No payloads are loaded on the prim and any descendant prims with payloads.

Example: In any building environment, such as the previous church example, there may be detailed items that you don't need to see to address a particular lighting task. Load None gives you an easy way of turning off loading for entire sections of your scene that are not in your area of focus. Aside from geometry, this could be any kind of data that you don't need, such as animation data.

Empty

the default value, or click until it resets to this value

If no rule is assigned and the prim has a payload, then the behavior depends on the load rules on the prim's ancestors and descendants.

For example, load the payload if any of the following is true:

- there are no load rules on the stage
- there are no load rules on any of the prim's ancestors
- the closest* ancestor with a rule, has the Load All rule
- the closest* descendant with a rule, has a Load All or a Load One rule

The payload is not loaded if:

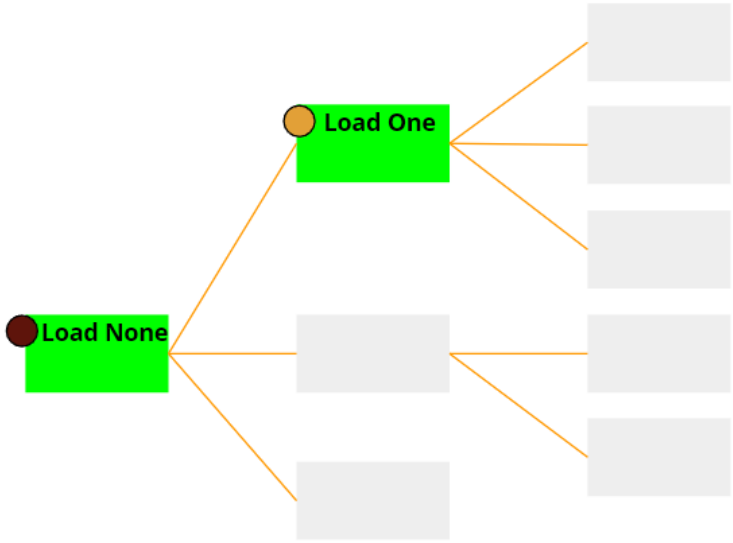
- the closest* ancestor with a rule, has a Load One or Load None rule.

(*closest can be any number of generations away, but the intermediate generations have no rule assigned)

Cascading Payload Rules to Refine Prim Loading

In real-world production scenarios, it may be preferable that certain parts of a scene should not be loaded to conserve memory. However, specific sections might still be required for a particular shot, sequence, or operation. With USD, child prims can override their ancestors' load policies, giving artists and developers precise control to optimize scenes for their specific requirements.

To achieve this, you can apply rules at different levels of the hierarchy to include or exclude specific prims as needed. For example, the tree below shows the effect of combining two rules.



A combination of payload rules on a simple graph. The leftmost node has a Load None rule but if present the payload would be loaded as there is a downstream Load One rule.

Activating and Deactivating Prims

When managing a scene, there are times when you need to toggle a prim's presence to remove it from the scene.

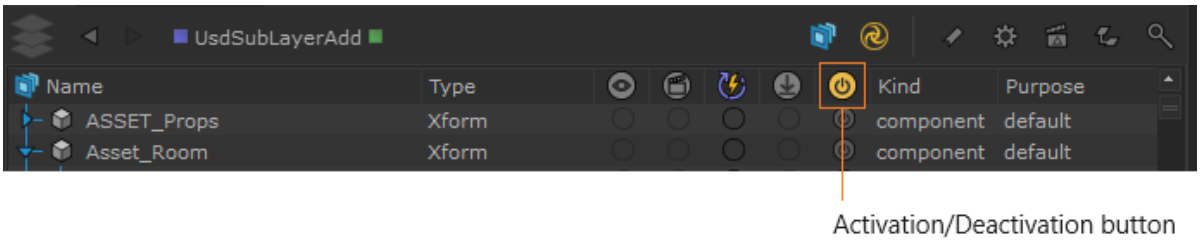
USD does not offer a way to delete primitives, as it is built for non-destructive editing. However, you can simply deactivate prims so they are hidden from the stage and have no effect on the result, removing them from stage composition.

The working set option in the Scene Explorer is a temporary action that will allow to toggle on/off per prim level. However, be aware that if these changes are not reflected in the usda file (i.e. incoming assets or toggled within Katana's node), then it won't actually be exported as part of the document.

Note: Since the "active/inactive" state is a Property in USD, you can also modify the active state using a UsdActiveSet node, which will add this property in Katana to any prim you'd like to remove.

Example: You may often have multiple layers contributing to the final scene. Activation/deactivation enables non-destructive overrides in more influential layers. For instance, a prim present in a base layer can be deactivated in an override layer, changing the final scene without altering the original content.

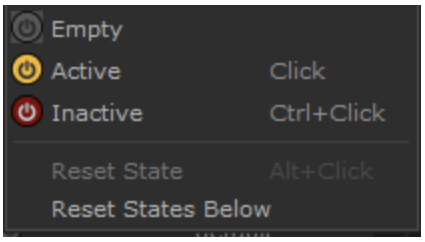
You can enable selective pruning by activation and deactivation using the Activation button in the Scene Explorer toolbar.



Click the power style button to enable activation/deactivation.

With the Activation column toggled on, you can easily deactivate and re-activate prims temporarily using the button in the column.

Click on the button or right-click to open the context menu.



With stage activation on, prim activation can have one of four states:

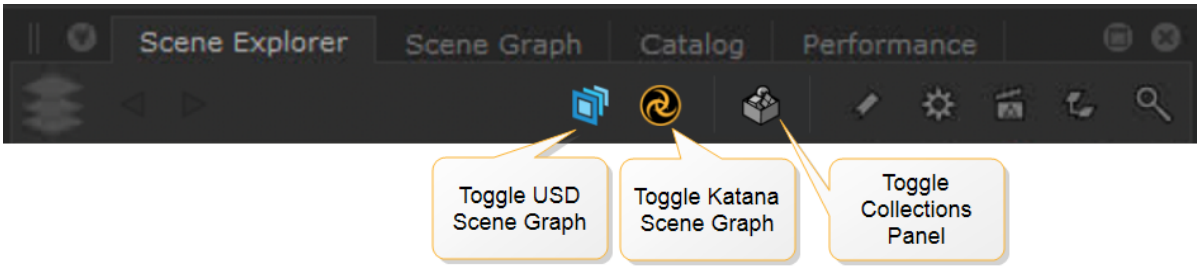
Status	Icon	Info
Empty (Default)		No active state has been assigned to the prim by the Scene Explorer, and the prim is active in the stage.
Inactive Set by USD or node		No active state has been assigned to the prim by the Scene Explorer, and the prim is inactive in the stage.
Active Click to toggle		The prim is set to active by the Scene Explorer. All the descendants are active unless otherwise deactivated.
Inactive Ctrl + click		The prim has been deactivated by the Scene Explorer, its descendants are not included in the stage.

Note: There may be other combinations due to the interplay of the Scene Explorer Active column control with the USD, but inactivity is essentially indicated by a light-grey power symbol. A color-filled circle means that the active status is being controlled by the Scene Explorer.

Collections in the Scene Explorer

There is now a panel above the scene graphs, showing the **Collections** in the project. This is collapsed by default.

Use the three selection buttons at the top of the Scene Explorer tab to display or hide the USD Scene Graph, or/and the Katana Scene Graph, or/and the Collections Panel.

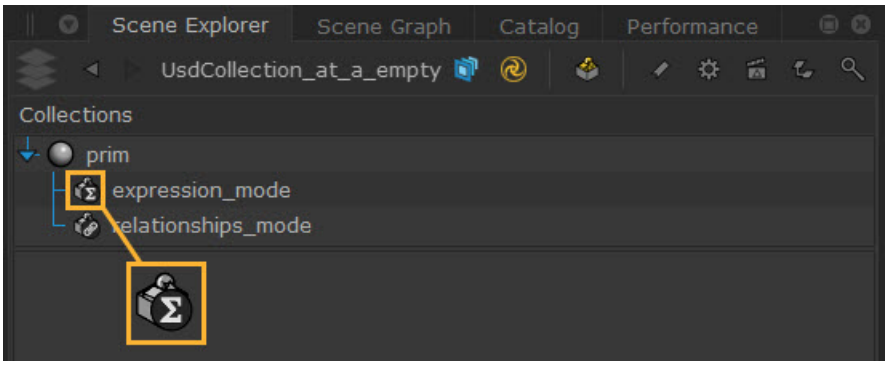


If a panel is hidden, its icon is greyed-out.

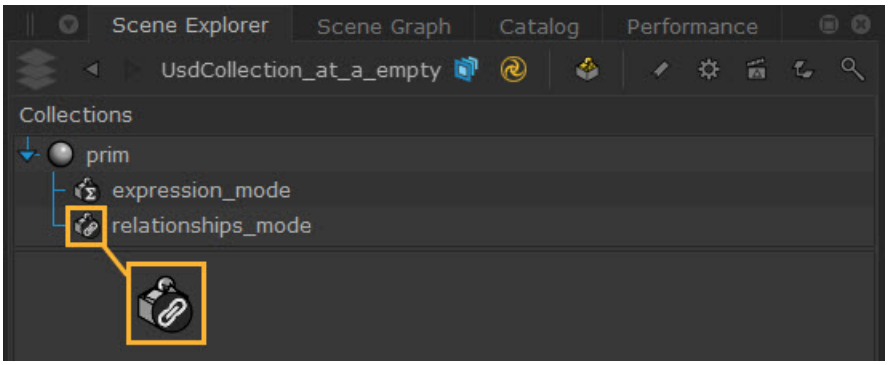
The **Collections Panel** is split into Katana and USD sections. Hiding the Katana or USD section of the main Scene Explorer tab by clicking the USD Scene Graph and Katana Scene Graph toggle buttons also causes the respective section of the Collections panel to hide.

Collections Panel

Expression mode collections set by a membership expression appear in the Collections Panel with the icon:

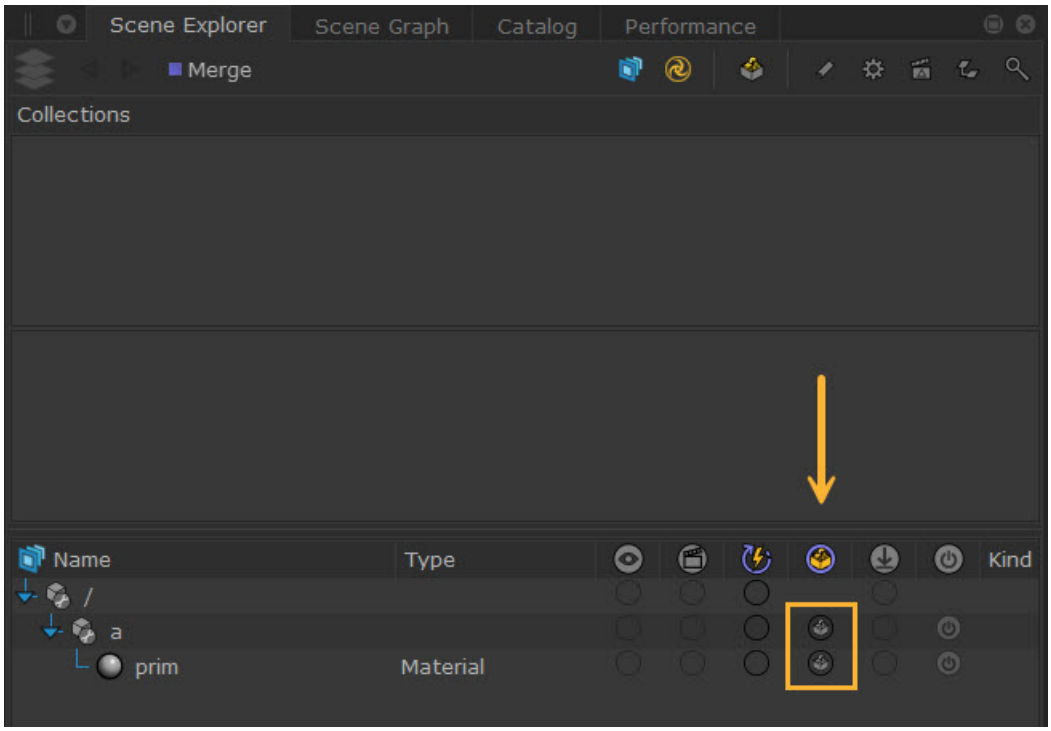


Relationship mode collections set by specifying includes/excludes appear in the Collections Panel with the icon:

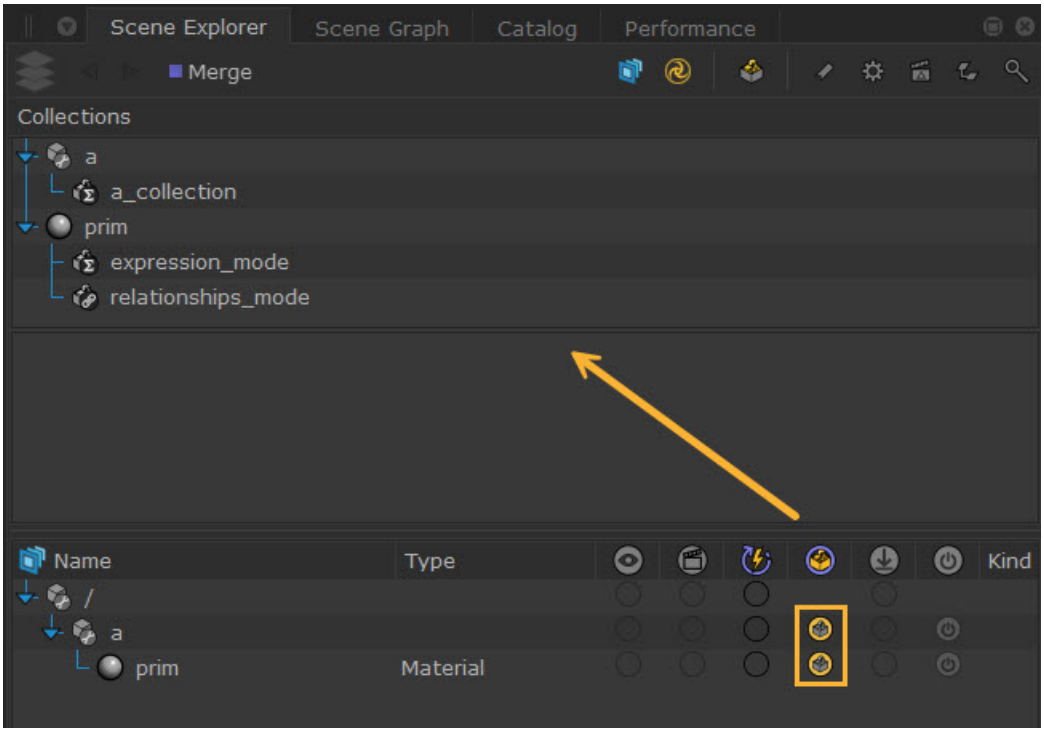


Using the Collections Working Set

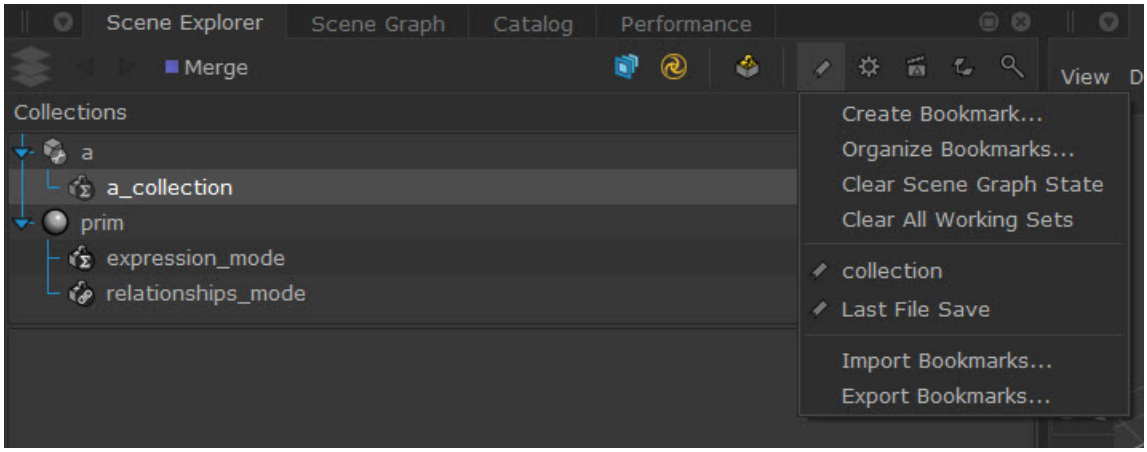
The Collections working set shows which locations have a collection on them.



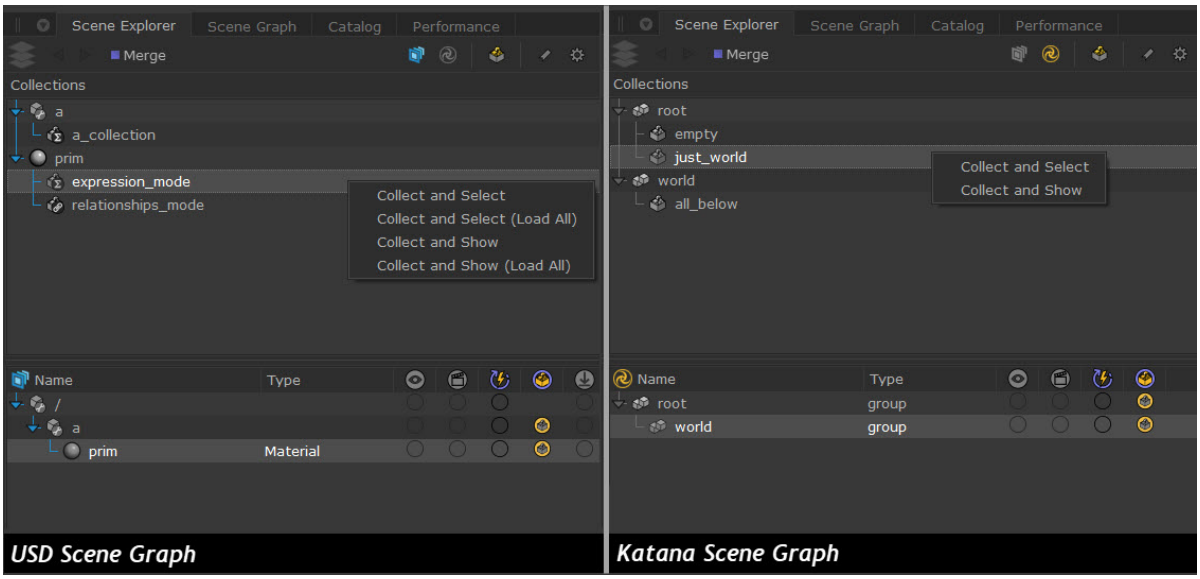
- To view which collections are available, click the **Collection icon**  next to a location in the Collections working set column. Once selected, the collection is displayed in the Collections panel above.



• Any selected collection in the working set can also be saved as a bookmark.



Selecting and Showing Collections



- Right click on a collection in the Collections panel and choose **Collect and Select** - this expands the branches and automatically selects them in the Scene Explorer.
- Right click on a collection in the Collections panel and choose **Collect and Show** - this displays all the paths contributing to that collection in a new window.

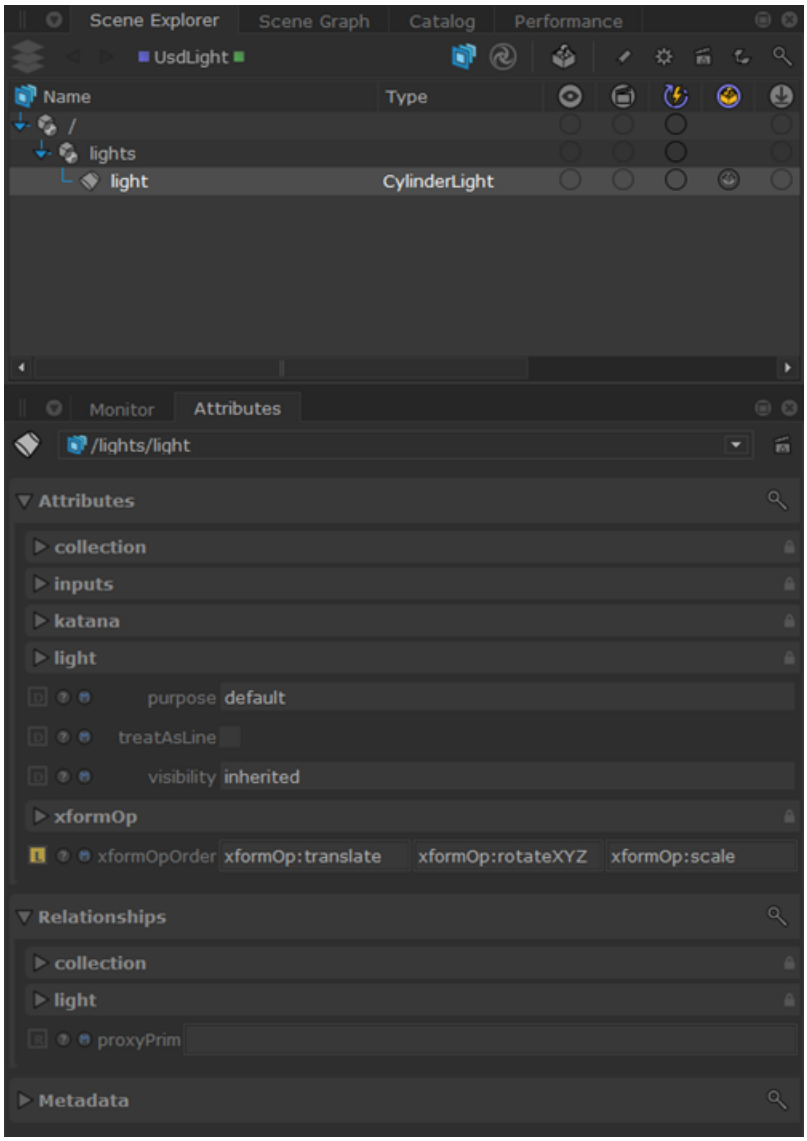
Only applicable for USD Collections

- Right click on a USD collection and choose **Collect and Select (Load All)** - this loads its Payloads while expanding the branches and automatically selects them in the Scene Explorer.
- Right click on a collection and choose **Collect and Show (Load All)** - this loads its Payloads while displays all the paths contributing to that collection in a new window.

For more about Collections, see Pattern-Based Collections with USD.

Inspecting USD Properties Using the Attributes Tab

The **Scene Explorer** tab integrates seamlessly with the **Attributes** tab for inspecting USD properties. As with the Katana Scene Graph, click on any location in the USD Scene Graph to view the properties in the Attributes tab.



Click on a scene graph location to view the properties in the **Attributes** tab.

- **Attributes** - Attributes store typed values linked to a prim, defining properties like appearance, transformation, and behavior. They can be either dynamic or hold constant values, and each has a specific data type such as float, string, or vector. Attributes can be set using UsdAttributeSet.

> **Tip:** You can drag and drop an attribute from the Attributes Tab into the UsdAttributeSet node's Parameters to have it adopt the attributeName, attributeType, and value.

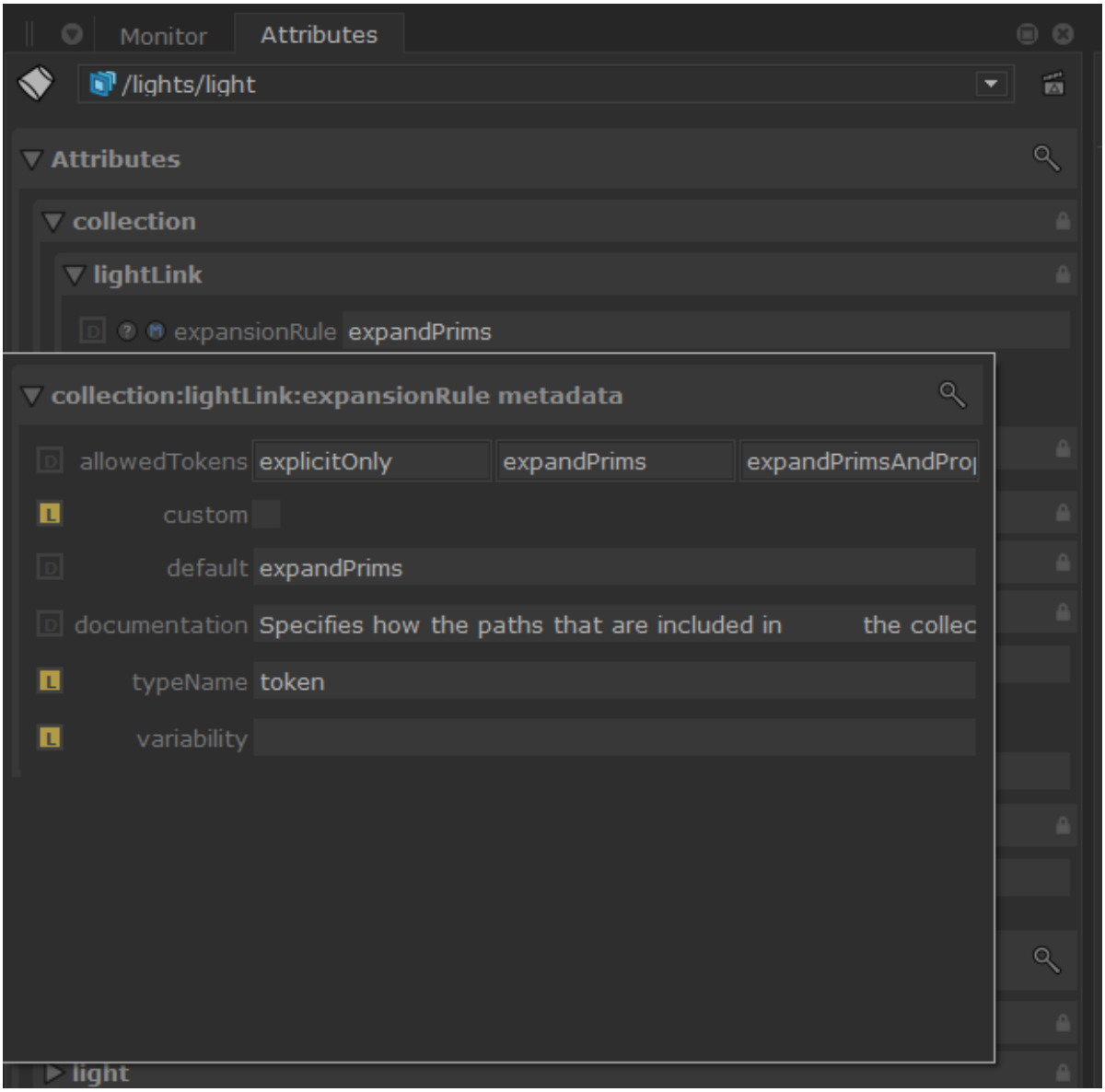
- **Relationships** - In USD, relationships establish direct connections between prims, allowing for referencing other prims or properties within the scene graph. Relationships can be set using UsdRelationshipSet.

- **Metadata** - This includes any prim metadata you have added such as apiSchemas, documentation, specifier and typeName.

You can also view the metadata for any of the properties in the Attributes and Relationships sections by clicking the metadata button  next to each property.

Tip: Metadata can be set using the UsdMetadataSet node.

Note: Learn more about Metadata in the OpenUSD Glossary.





Note: For more information on USD properties, see Attribute and Relationship in the OpenUSD Glossary.

Manipulating USD Data Natively

Editing USD Properties

UsdAttributeSet - The same as AttributeSet but for native USD. For USD, attributes are one type of Property that a prim may have.

UsdRelationshipSet - Relationships are the other type of Property a prim may have. They are specifically Properties that point to other Properties or prims, eg. Material binding, light linking etc.

UsdMetadataSet - Used to assign metadata to a specified prim.

UsdPrimvarSet - Used for creating primvars. Primvars are data attached to geometry that can interpolate a value (eg. color) over surfaces or volumes to achieve variation. The interpolation type can be chosen, eg. faceVarying - varying values on each face, vertex - values are interpolated between each vertex.

UsdSchemaSet - Can be used to assign a schema API to a prim.

Manipulating USD Data in the Scene

UsdTransformSet and UsdTransformEdit - Transform USD prims.

UsdVariantSet - USD layers can contain variations (eg. geometry/materials variants). This node is used to select a variantSet (eg. character suit colour) and variant (eg. red) for the specified prim.

UsdActiveSet - Used to define what is included in the stage composition. Inactive prims are removed from the stage in a non-destructive way so they are not visible in the Viewer or renders.

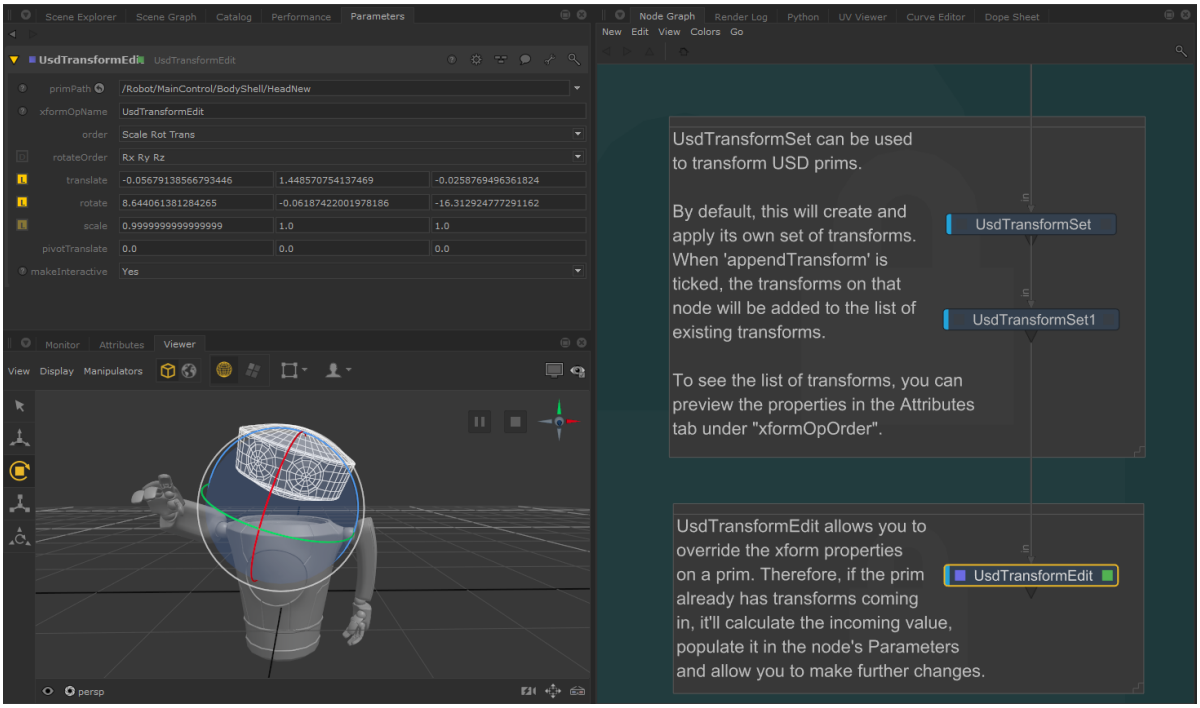
UsdToKatana - Used to bring certain elements from USD into Katana so they can be manipulated eg. to use NME, to inherit a light into a GafferThree to make edits.

Native USD Transformations

In USD, **transformations** are used to position, orient, and scale prims in 3D space.

To perform transformations in Katana, you can use the UsdTransformSet node to set transformations, and UsdTransformEdit to overwrite existing transformations. Other nodes also allow you to edit their transformation values directly in its parameters, for example UsdXform, UsdCamera, and UsdLight.

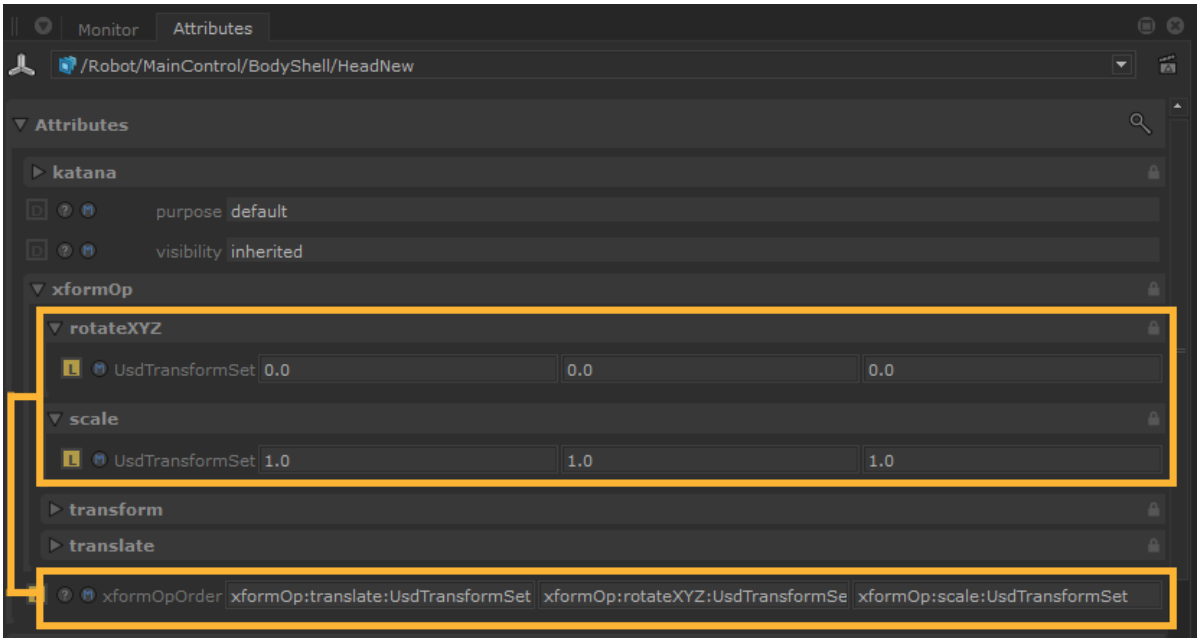
> **Tip:** The scenes used in the examples below can be found in Katana under **Help** > **Example Projects** > **Native USD** > **Manipulating USD Data**.



When editing a scene with transformations applied, you'll see that USD stores transformation data via **XformOpOrder**. One way to think about xformOpOrder is that it describes a series of transformations to apply.

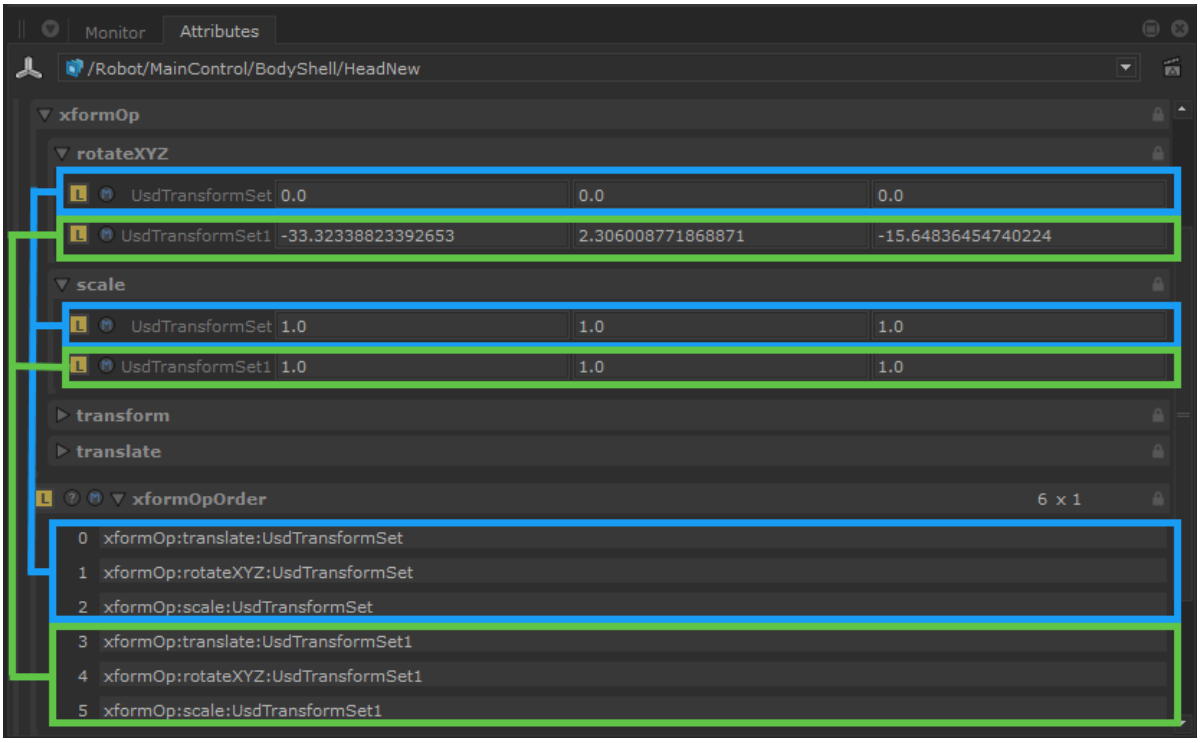
When transforming via the UsdTransformSet, the rotateXYZ, scale and translate values are stored in the **Attributes** tab and referenced within the **xformOpOrder** property.

This data is visible and editable in Katana in the **Attributes** Tab:



On the UsdTransformSet node, the **appendTransform** parameter can be toggled to drive how these values are stored:

- When disabled, the xformOpOrder will store the values set on that node, and discard any previous/incoming transformations on that prim. You will end up with one set of transformations under the xformOpOrder, as seen in the image above.
- When enabled, the xformOpOrder will "append" the transformations set on that node, and add it to the list, in addition to any previous transformations that had been set on the prim. This can be seen in the image below.

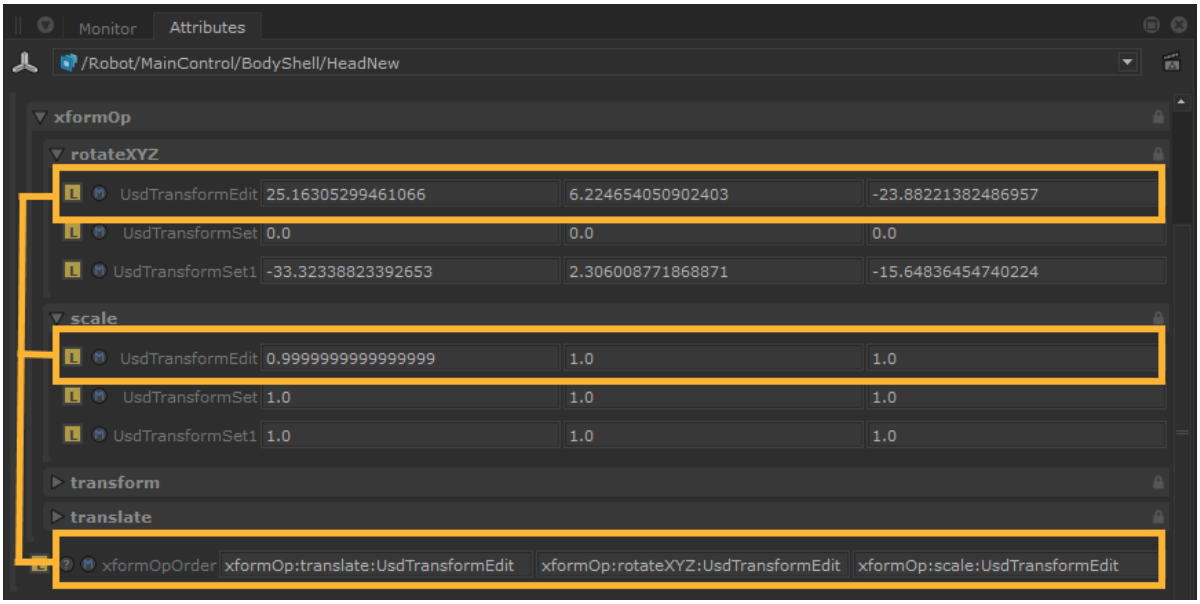


Note: As of Katana 7.5, the attributes created by the UsdTransformSet node are authored in scale, rotate and translation components, rather than matrices (as it was in Katana 6.5 and Katana 7.0). When **makeInteractive** is set to **No**, the **asMatrix** parameter is still available as an option to set these attributes as a matrix.

Using UsdTransformEdit

In contrast, the UsdTransformEdit node will identify incoming transformations on a prim, and automatically populate the transformation parameters on the node itself. When modifying the prim's transformations from the node interactively, it will override the **xformOpOrder** to only store the values set on the node.

The snapshot below shows that the prim contains multiple transforms from upstream UsdTransformSet nodes. However the UsdTransformEdit in **edit** mode will override to **xformOpOrder** to only take its own transformations into consideration.



To use the UsdTransformEdit, you can follow the same steps as UsdTransformSet node, with one additional step. After specifying your **primPath,** you will also need to fill in the **xformOpName** field, which will populate the translate, rotate and scale with its values. You can then use the Manipulators to change the values, and this will overwrite them. See Viewer Manipulators for Native USD below.

Note: The UsdLight node, when set to **edit** mode will follow the same behaviour as the UsdTransformEdit node and override the transformation properties the same way.

Viewer Manipulators for Native USD

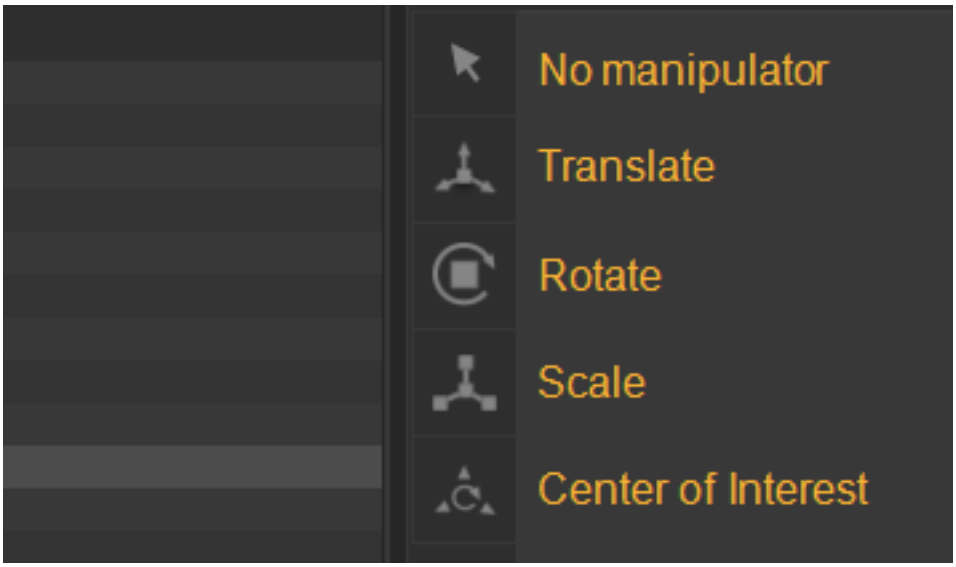
You can use Viewer transform manipulators against USD native scene locations, via various USD nodes: UsdTransformSet, UsdTransformEdit, UsdCamera, UsdLight, and UsdXform.

For example, using the UsdTransformSet node, you can allow interactive manipulation in the Viewer tab by following these steps:

- Connect a UsdTransformSet node to your scene.
- Specify the location to be transformed in the **primPath** parameter.
- Tick **appendTransform**, if you want to add the transformations to the list of operations in USD. In the example below, the robot had pre-existing transforms, that we can see applied when we toggle that option.
- Make sure the **makeInteractive** parameter in the UsdTransformSet node is set to **Yes**, which it is by default.

Note: Interactive transformations against USD native locations are available exclusively when the mode parameter is set to **Single Path**. When set to multiple locations, interactive transformations through the Viewer are currently unavailable.

- Select the prim in the **Viewer** or **Scene Explorer**.
- You can then use Viewer manipulators through the UI in the **Viewer** tab, or use shortcuts to Translate '**W**', Scale '**E**' and Rotate '**R**'.



Note: The **Center of Interest** manipulator is currently not supported with USD native prims.

Bounding Boxes in USD Scenes

Bounding Boxes in the Viewer

The **Viewer** tab allows you to see bounding boxes on native USD geometry. By **default**, bounding boxes are shown for Prims meeting the following criteria:

- The Prim is a 'model' with valid authored **extentsHint** attribute.
- No descendants are Viewer-visible (according to scene graph expansion / Viewer Visibility).

Additionally, bounding boxes are drawn for *boundable* prims if they are *unloaded* (and so are excluded from Hydra rendering). Bounding boxes for unloaded prims have a subtle red tint, distinguishing them from those for loaded prims.



Note: Where **extentsHint** is authored for multiple purposes, Katana displays only those for geometry that would be rendered in a final render (namely 'default' and 'render'), since the visual representation is a proxy for final geometry.

Bounding Box Options

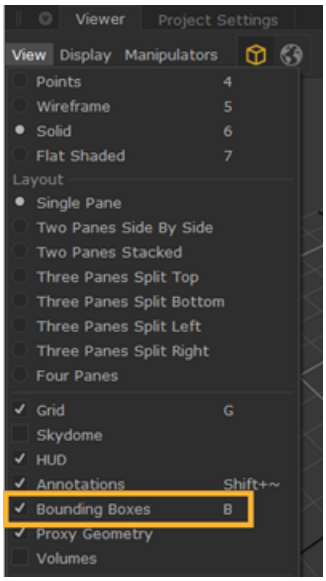
Display of bounding boxes may be customized (or disabled) by setting the following values for the `KATANA_VIEWER_PRIM_BBOX` environment variable, as detailed in the Environment Variables section of the Katana Developer Guide.

- **0**: No bounding boxes are shown.
- **1**: Default (as described above).
- **2**: Bounds are drawn for unloaded prims (not loaded prims), even if descendants are Viewer-visible.
- **3**: A combination of the two previous options (loaded Viewer-visible leaf prims, and all [visible] unloaded prims).

Viewing Bounding Boxes

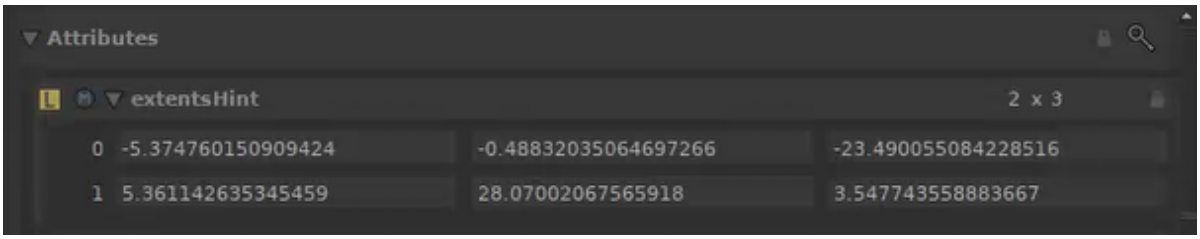
Bounding boxes are enabled by default. To turn bounding boxes on and off you can use the Viewer dropdown.

In the **Viewer**, find **View** > **Bounding Boxes**. Toggle the checkbox to enable or disable them.



Bounding Boxes in the Attributes Tab

In the **Attributes** tab, you can see the USD prim's bounding box data and use the **extentsHint** attribute to add bounding box data to higher up locations.



Note: We expect for the case of Payloads, that the extentHint parameter is placed on the location where the payload state is set, in order to avoid "loading" it to obtain their bounding box information.

Pattern-Based Collections with USD

Pattern-Based Collections (PBC) was introduced in USD 23.11 and is now fully integrated in Katana.

Similar to Katana's pre-existing Collections and CEL, USD's **Pattern-Based Collections (PBC)** allows you to:

- Use **Paths**, **Collections** and **Custom** statements as inputs.
- When multiple statements are set, use additional options to determine the relationship between them, such as **Union**, **Difference**, **Intersect**, and **Complements**.

Paths	These are explicit lists of scene graph paths. If you drag and drop locations from the Scene Explorer tab onto the **Add Statements** area of the **PBC widget** you are automatically given a **Path statement.**
Collections	A pre-defined named collection of Scene Explorer 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 UsdCollection node.
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.

Using Patterns

For example, rather than specifying rules for paths to include and exclude in the collection, you could provide a **Path statement**. This can contain patterns, operators, and references to other expressions, capturing the set of prims you want to match.

A **pattern** is a sequence of characters that follow a set of syntax rules, with the option to use glob-style **wildcard** characters.

Useful wildcard characters:

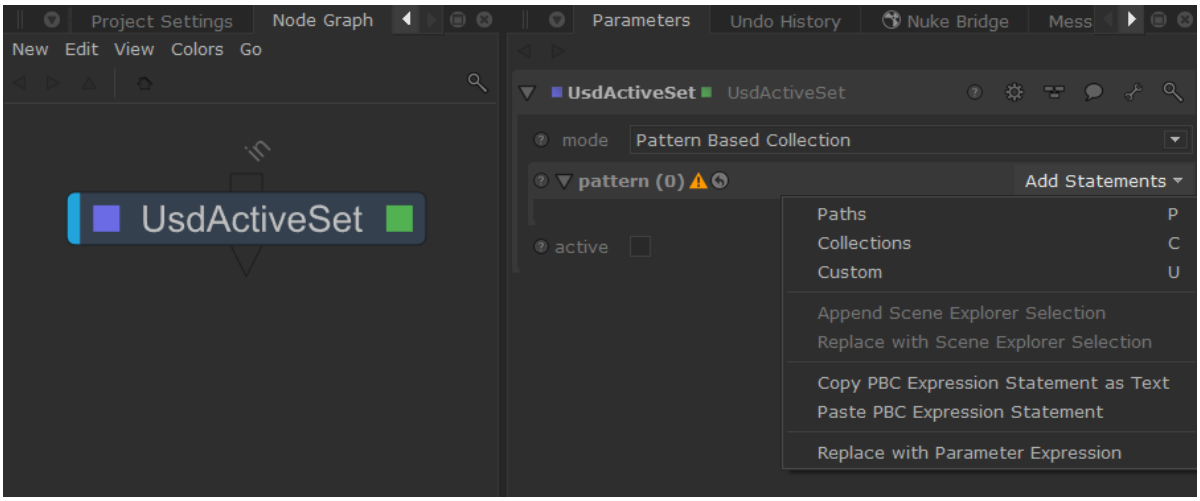
Character	Function
*	Matches zero or more characters, but does not match path or property separators.

Character	Function
?	Matches exactly one character, but does not match path or property separators.
//	Matches any path hierarchy, allowing you to match a sequence of paths of any length.

Note: For a full overview of supported Pattern-Based Collection Expressions, see OpenUSD's documentation: Collections and Patterns.

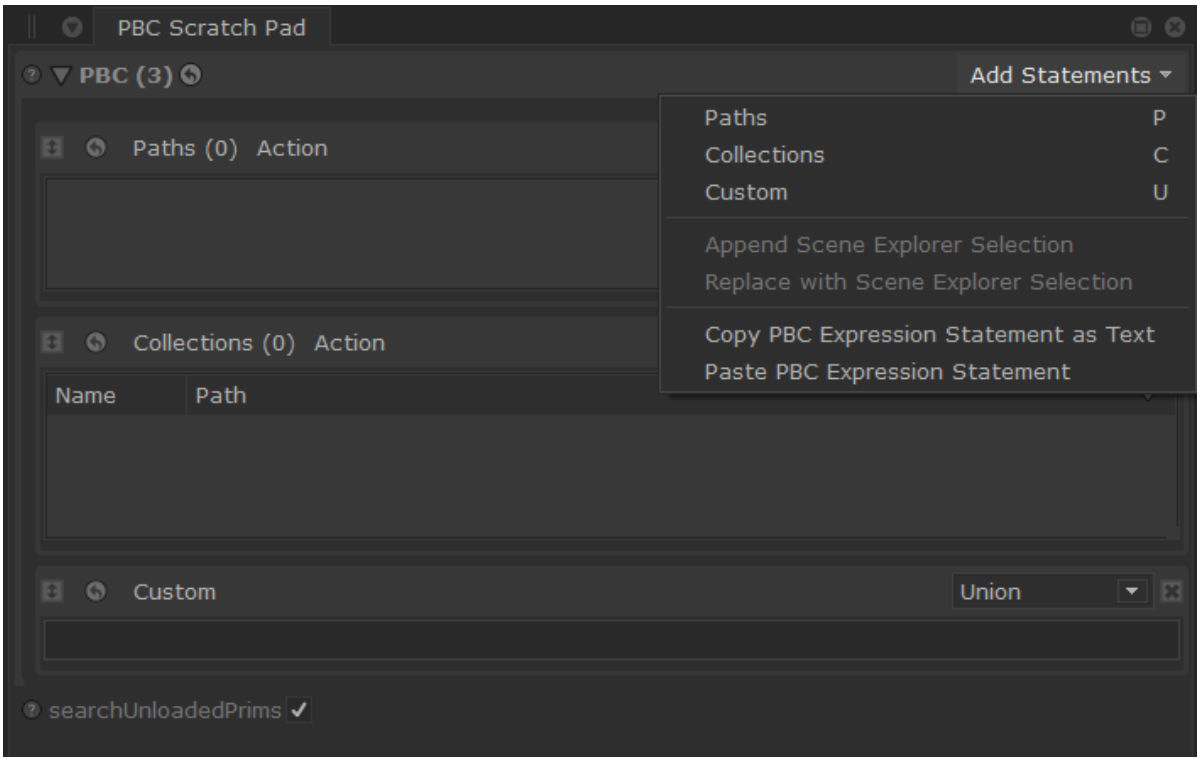
Using PBC in Katana

- The **PBC widget** is available in the relevant **USD nodes** in Katana, such as the UsdActiveSet, the UsdVariantSet, the UsdRelationshipSet nodes and many more.
  This allows you to target multiple prim paths at once using these nodes, using a set of rules, (such as Paths, Custom expressions or Collections).



Warning: PBC widgets no longer accept relationship-mode collections (membership determined by 'includes' and 'excludes' lists).

- Additionally, there is a **PBC Scratch Pad** tab that can be used in a similar way as the CEL Scratch Pad tab. When **searchUnloadedPrims** is enabled, the pattern is matched against collapsed primitives in the Scene Explorer. When disabled, the result only shows expanded locations.



- You will also need a UsdCollection node to define and modify USD collections - see reference guide page UsdCollection.
- You can see a list of the CEL and USD (PBC) Collections in your project in the Collections panel at the top of the Scene Explorer and via the Collection working set indicators. See Collections in the Scene Explorer.

PBC Scratch Pad - Example

We can use the **PBC Scratch Pad** tab to 'sketch' out patterns, collections and custom expressions, in a similar way as the CEL Scratch Pad tab.

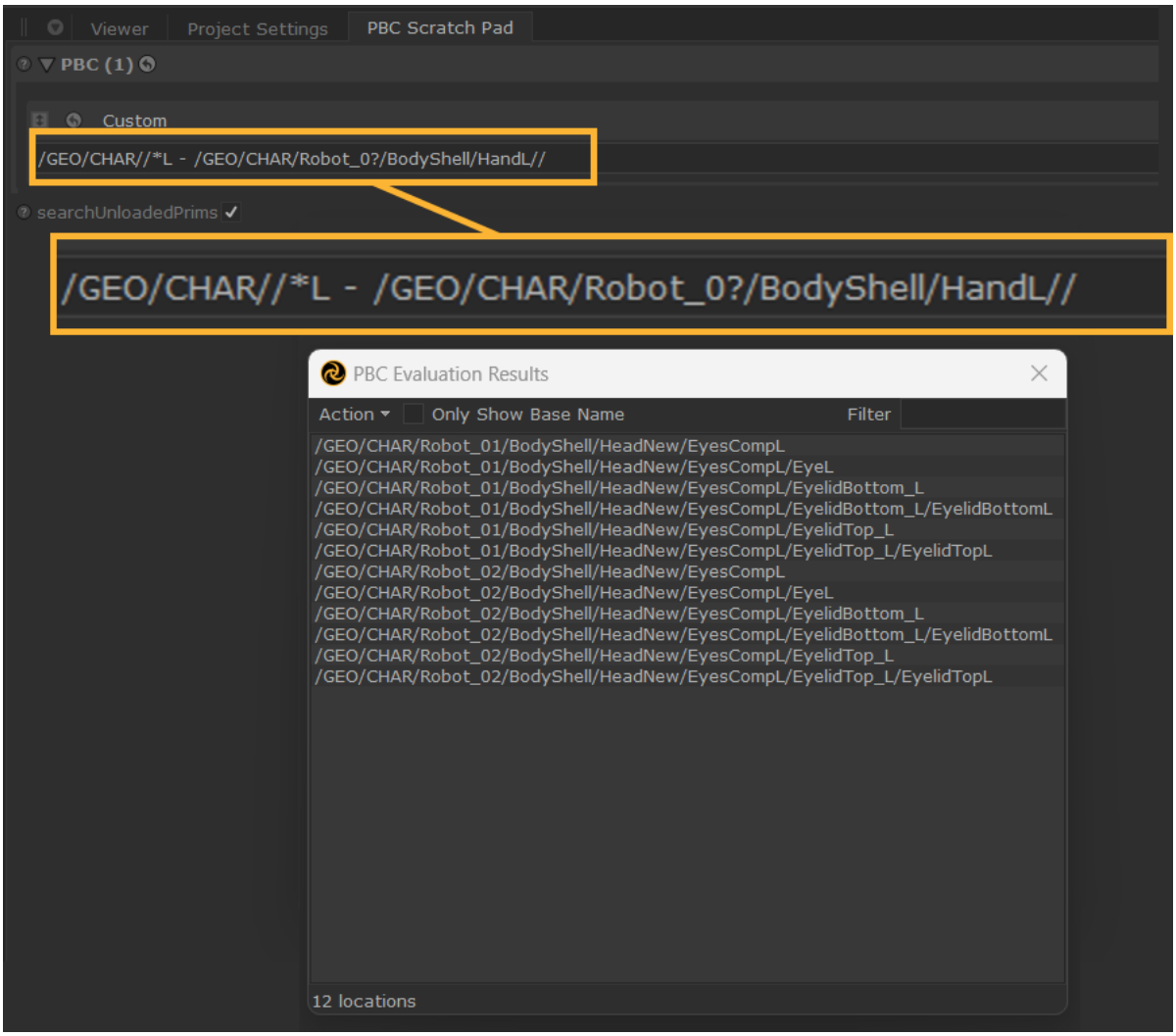
An example workflow would be to use the PBC Scratch Pad to write and test Custom expressions for use in other nodes. The option 'searchUnloadedPrims' can come in handy here.

In the example shown below, we are using one of the example projects that can be found under **Help** > **Example Projects** > **Native USD** > **Native USD Lighting Example**.

Using PBC, we can make a selection of all the components that are part of the left eye. Here, we are grabbing all the primitives that include the letter 'L' but excluding the ones that are part of the hands.

/GEO/CHAR//*L - /GEO/CHAR/Robot_0?/BodyShell/HandL//

Characters	Function
…//*L…	To include any primitive under /GEO/CHAR/ that has ‘L’ in its name.
…-…	Subtract the result of the following pattern from the previous pattern.
…/Robot_0?/…	Include any primitive and its children if they match the pattern minus the last letter/number. In this case: Robot_01 and Robot_02.
…HandL//…	Include anything under the HandL primitive.



Native USD Layer Export

Katana USD Layer Export enables you to **export Native USD data to a USD layer**. This allows the export of entire layers and stages including geometry, cameras, materials, assignments, and lights.

To **export a specific set of nodes** from your node graph, there are two ways you can do this:

- Use UsdLayerDefine + UsdLayerExport to define the selection
- Use UsdLayerExportGroup to export as a group

In both of these cases, the same underlying nodes and workflows are used.

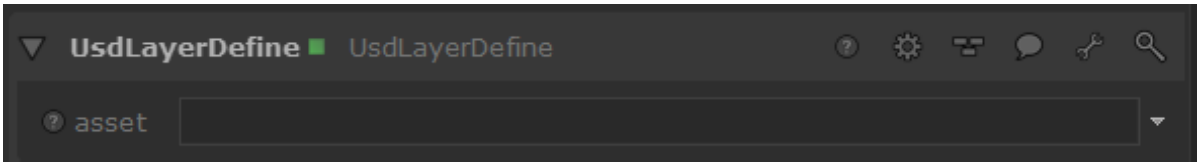
Or, to **export all upstream changes**:

- Simply use UsdLayerExport. In this case, you can use the following guide and skip steps 1 and 2.

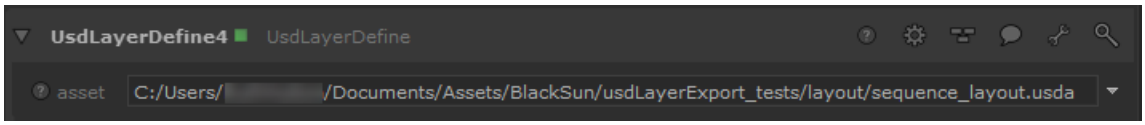
Using UsdLayerDefine + UsdLayer Export nodes

1. Set up a UsdLayerDefine node:
   Add this node to act as a start point for the exported layer. The UsdLayerExport node then captures the changes made between this start point and its second input. This node is required if you are exporting from a point where nodes exist above. If there are no nodes above your start point then this node is not required.
2. Specify USD layer in the **asset** parameter:

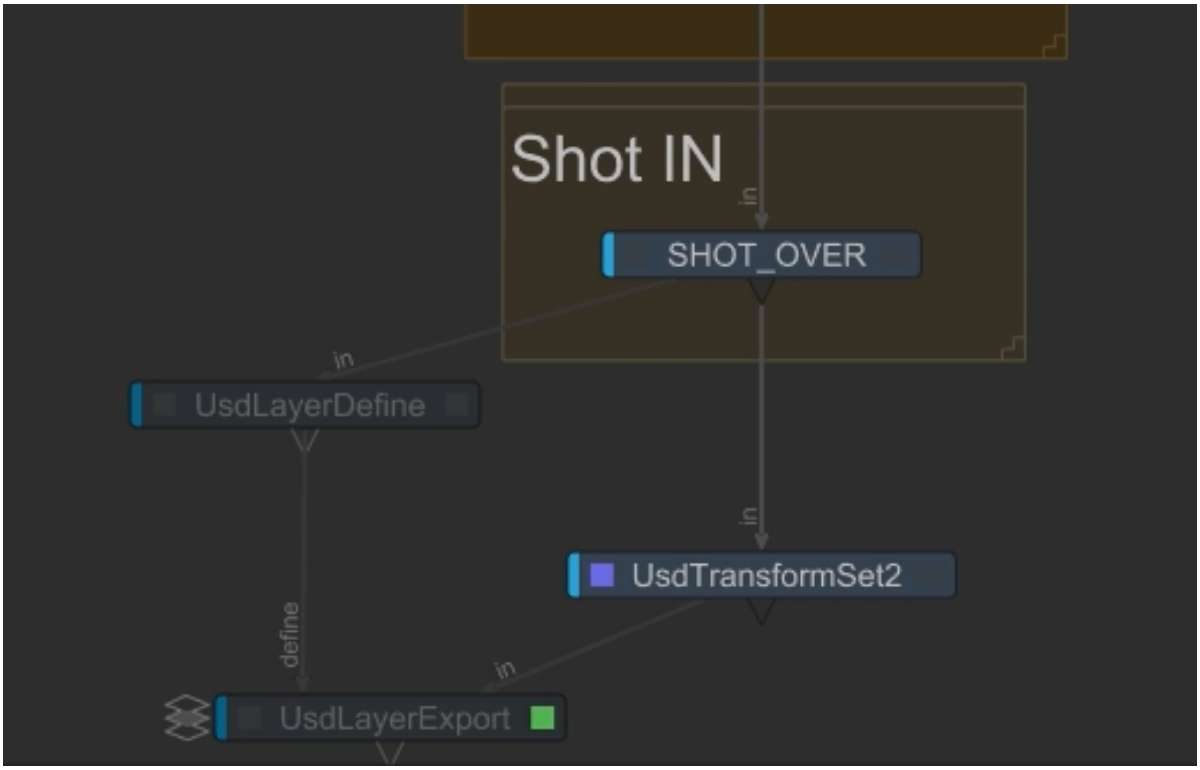
- If no layer is specified, then the changes will be exported as overs so that the exported layer only contains the edits made and does not replace other layers.

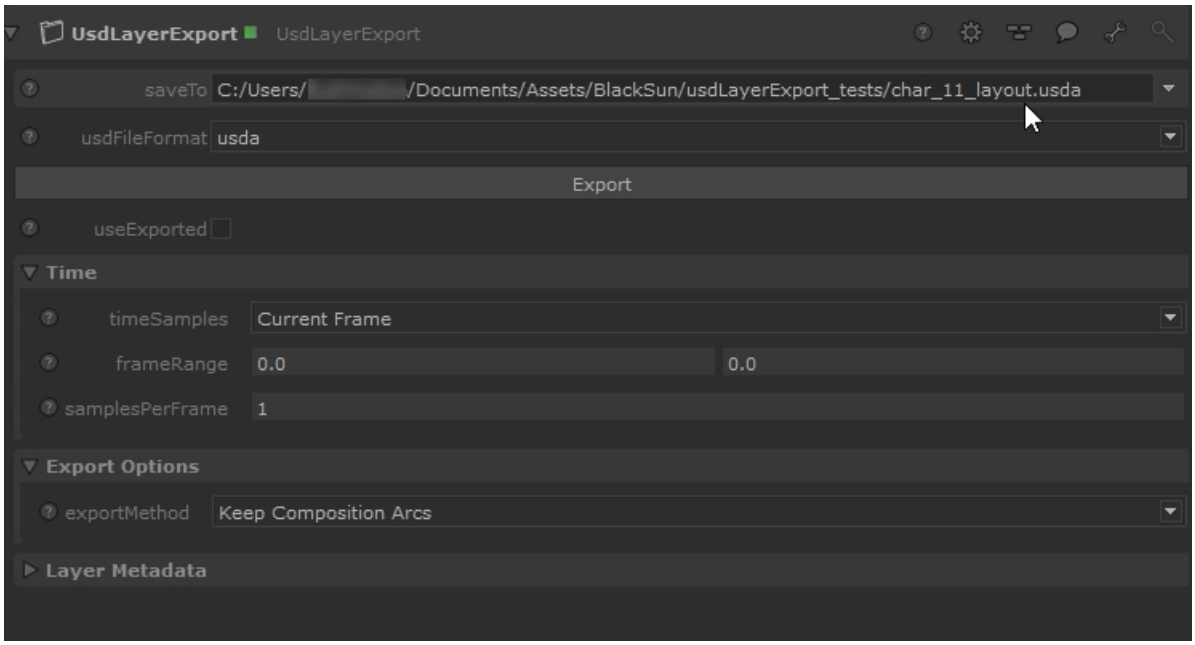


- If a layer is specified, then the exported layer will combine the specified layer with the changes made with the purpose of replacing the specified layer.



3. Add UsdLayerExport node:
   Connect this node to export Native USD data to a USD layer.

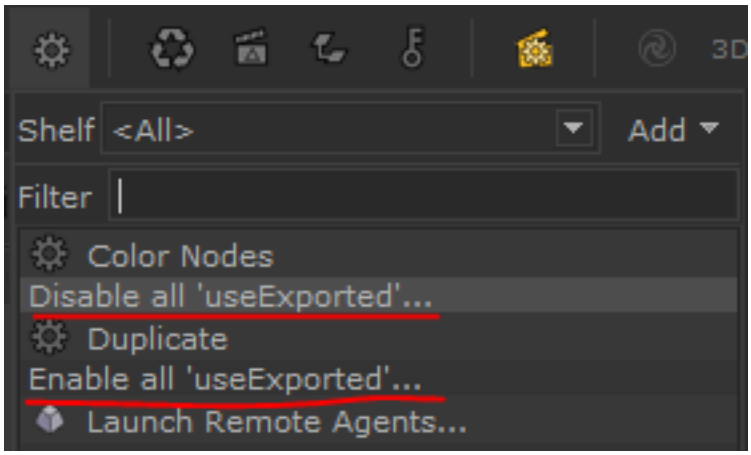




Note: Only changes using native USD nodes will be baked.

You can load in the exported USD layer by using the **useExported** toggle. This is particularly helpful for performance improvements or comparing against an exported layer. This will effectively bring the baked layer in as a sublayer. When multiple UsdLayerExport nodes exist in a scene, you can use the shelf tools -

Disable all 'useExported'... and **Enable all 'useExported'...** - to enable or disable the option across all nodes.



Multiple export options can be found, such as export methods (**Keep Composition Arcs**, **Flatten Sublayers** and **Flatten All**) to tailor the USD layer file to your needs, as well as other parameters including the ability to choose a desired file type, time related settings, export options and several options for setting up layer metadata.

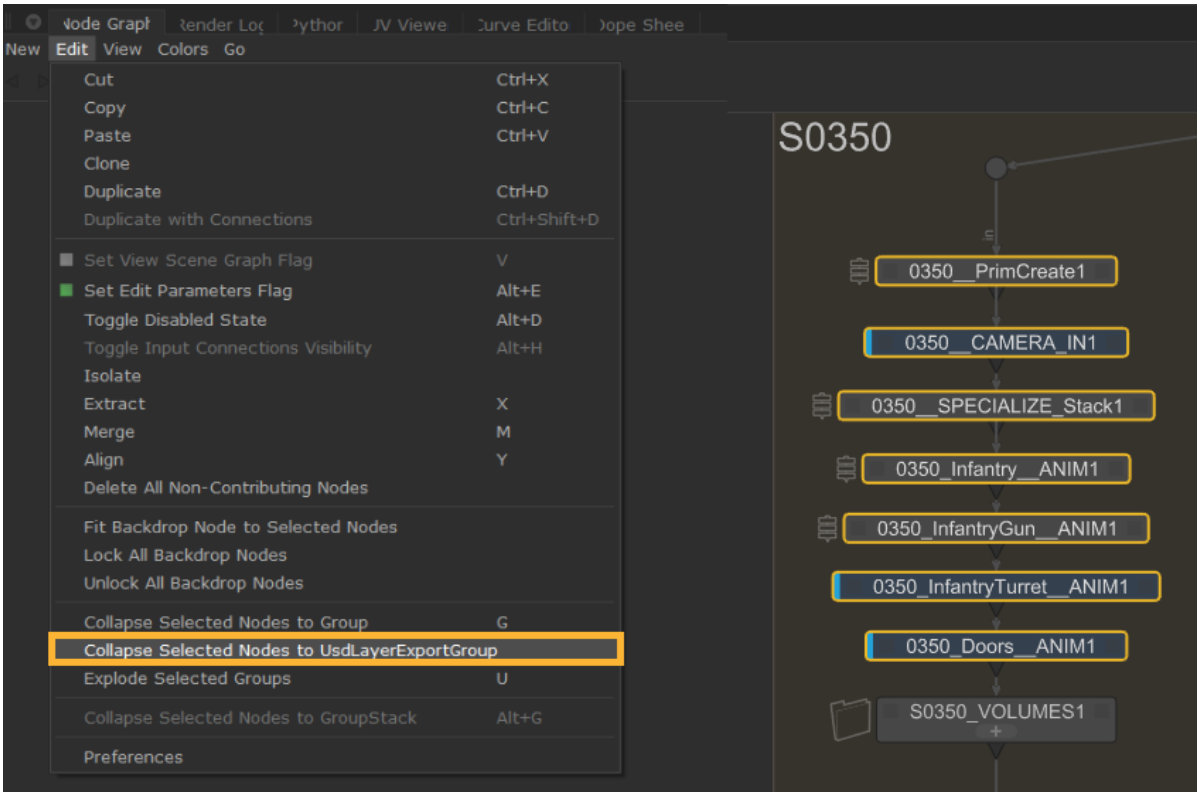
Note: You can right-click UsdLayerExport nodes to bake without having to open the parameters.

4. Select **Export** to create the export with the options you have chosen.

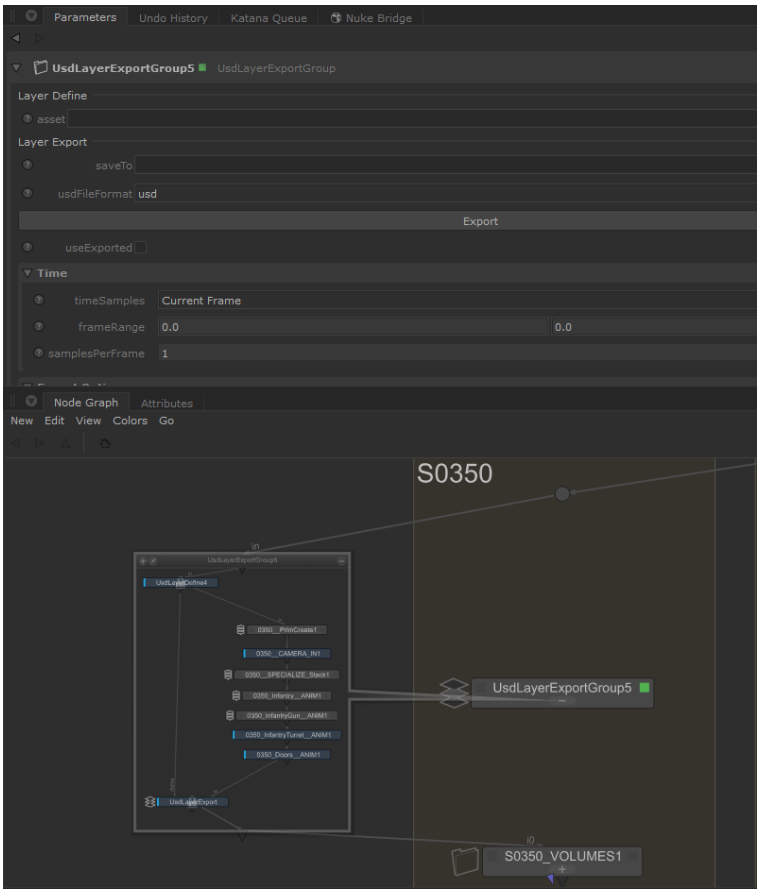
Using the UsdLayerExportGroup node

Using the UsdLayerExportGroup node, you can also select multiple nodes and add them to the Group ready for export without having to do the set up manually:

1. To do this, select the nodes and go to **Edit** > **Collapse Selected Nodes to UsdLayerExportGroup**



2. The Parameters for both the UsdLayerDefine and UsdLayerExport nodes are now available on the UsdLayerExportGroup's parameters, where you can adjust export options.



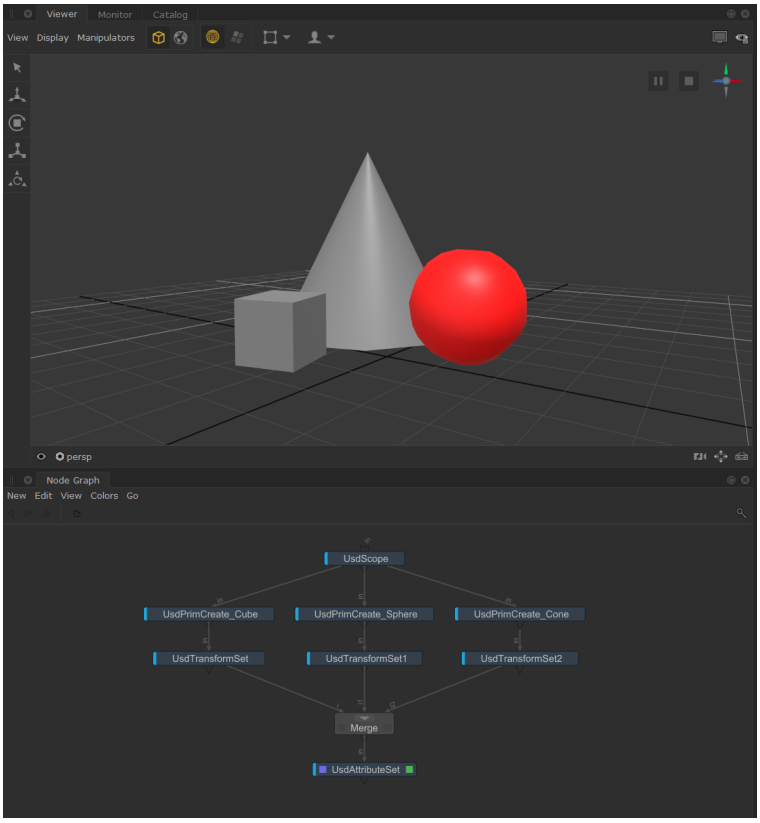
3. Select **Export** to create the export with the options you have chosen. Refer to the steps above for guidance on how to populate these parameters

Modifying using the UsdLayerExporter Plug-in

You can also use the **UsdLayerExporter Plug-in** to access the USD layer prior to exporting. This can be used for modification, validation, or performing any action on the layer in memory before exporting. More information is available in the developer guide under **Usd Processing Engine** > **UsdLayerExport**, with examples in: **<INSTALL_DIRECTORY>/plugins** and available through a new **KATANA_EXAMPLES** subfolder, **/UsdLayerExporterPlugins**.

USD Export - Practical example

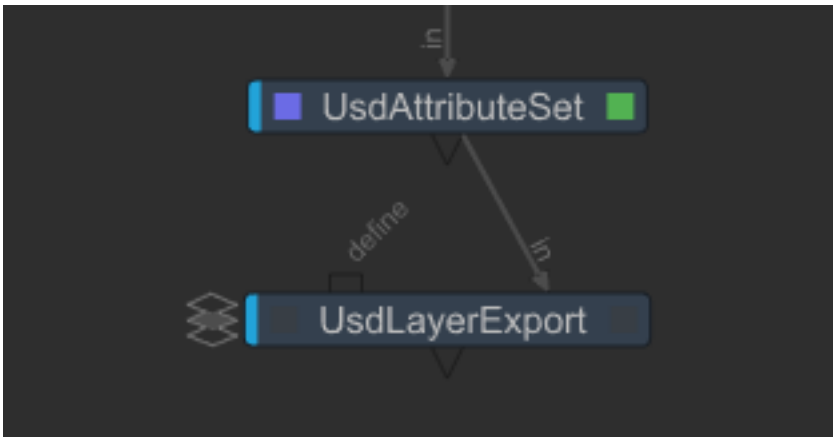
Here we have a basic example scene consisting of a cube, a sphere and a cone. We've moved them around in space and assigned a red color to the sphere using a UsdAttributeSet node.



Let's export this to a **.usda** file. Click through the steps below.

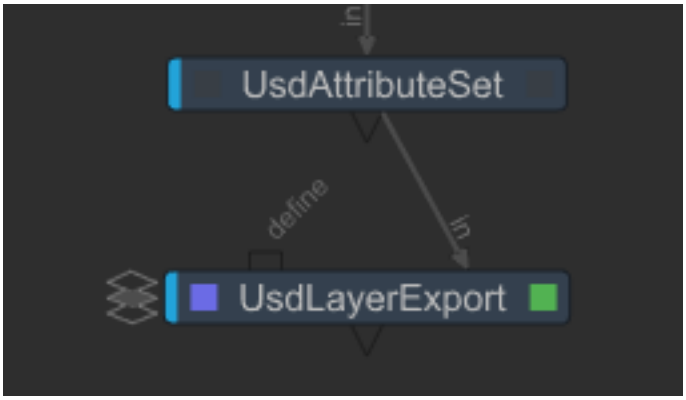
1 START

Start by adding a UsdLayerExport node and connecting it to the 'in' input downstream of the UsdAttributeSet node.



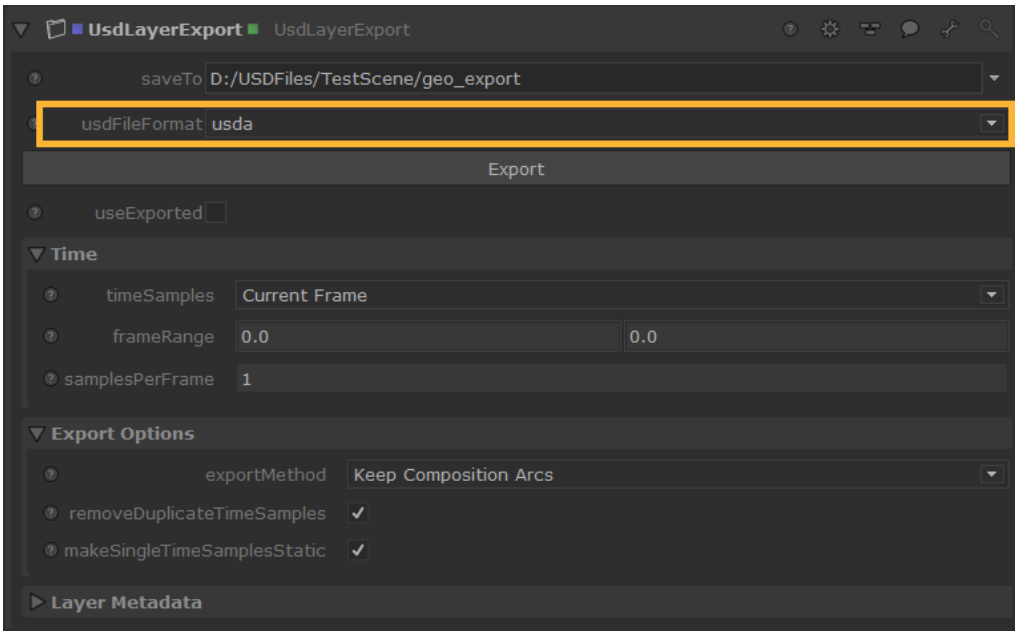
2

Set the parameter flag on the UsdLayerExport node by double clicking the node.



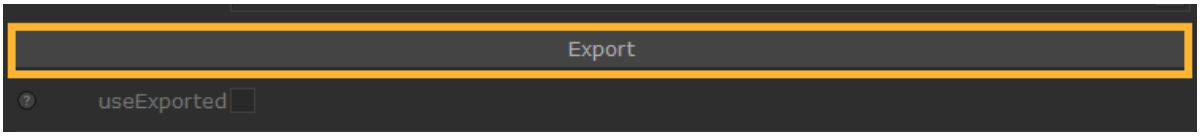
3

In the Parameters tab, specify a path as to where to save the USD file and set the **usdFileFormat** to **usda**. This will write the USD file in an ASCII format so we can open it in an external code editor. Since we don't have any animation or time sensitive data in our scene, we can keep the time options at default, meaning only the current frame will be exported.



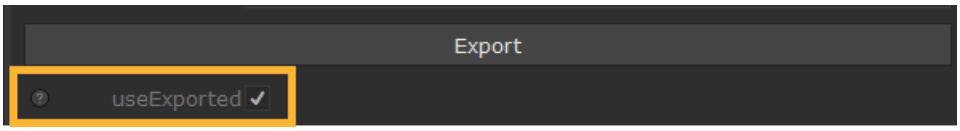
4

Click the **Export** button to write your USD stage to a **.usda** file. Depending on the size of your scene, a pop up might show, indicating the export is ongoing.



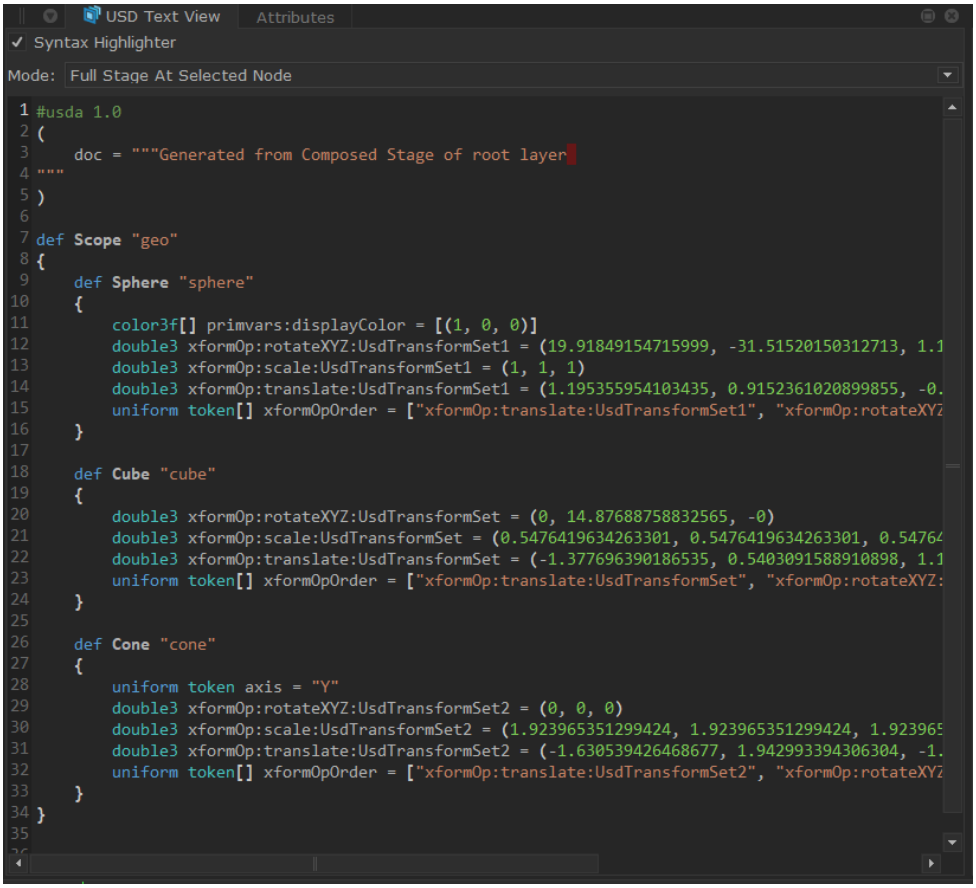
5

With the export finished, we can use the **useExported** parameter to load in the file we just saved out (similar to a UsdSublayerAdd node). This can be particularly useful for performance reasons.



6

We can now look at our freshly exported .usda file with Katana's USD Text View tab with its **mode** set to **Full Stage at Selected Node**.



USD Debugging

If something isn't behaving as expected, there are a few ways for you to inspect the scene and diagnose the issue.

The **Attributes** tab is already a great tool to inspect the scene, enabling you to go through each section of the Node Graph and identify changes.

Note: See Inspecting USD Properties Using the Attributes Tab.

However, if further information is needed, digging into the resulting **usda** contents will provide all the information we'd need to pinpoint any mistakes. For this you may find the **USD Text View** tab and **USD Error Handling** useful.

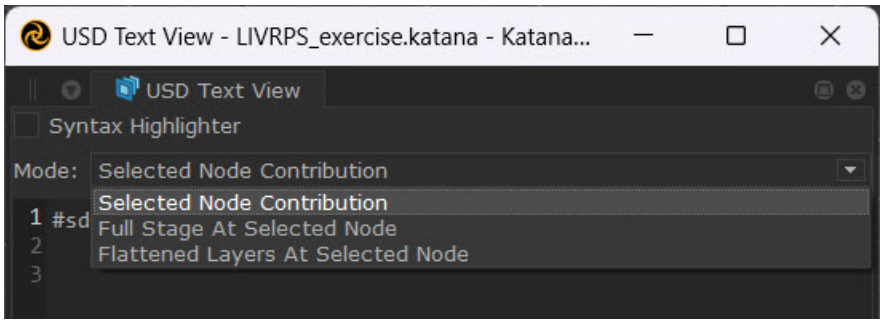
Note: See USD Text View and USD Error Handling.

USD Text View

When selecting a node, the **USD Text View** tab displays the USD code of the composed stage up to that node.

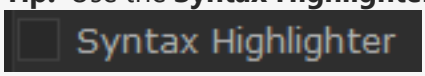
To view it, go to **Tabs** > **USD Text View**.

When you select the **Mode** dropdown, there are three viewing options, depending on what context you want to look at the node.



- **Selected Node Contribution** - This option will display only the contributions of the node selected in the Node Graph, excluding any description from upstream nodes.
- **Full Stage At Selected Node** - This option will display the entire scene description going up to the selected node, including information from upstream nodes.
- **Flattened Layers at Selected Node** - This option will display the incoming stage as flattened layers rather than the fully composed output. This will retain composition arcs in the text view, but apply sublayers.

Tip: Use the **Syntax Highlighter** checkbox above to turn syntax highlighting on or off.



This is disabled by default, to improve the performance of the tab. However, turning highlighting on can help with easily identifying the elements of the code and help you identify issues.

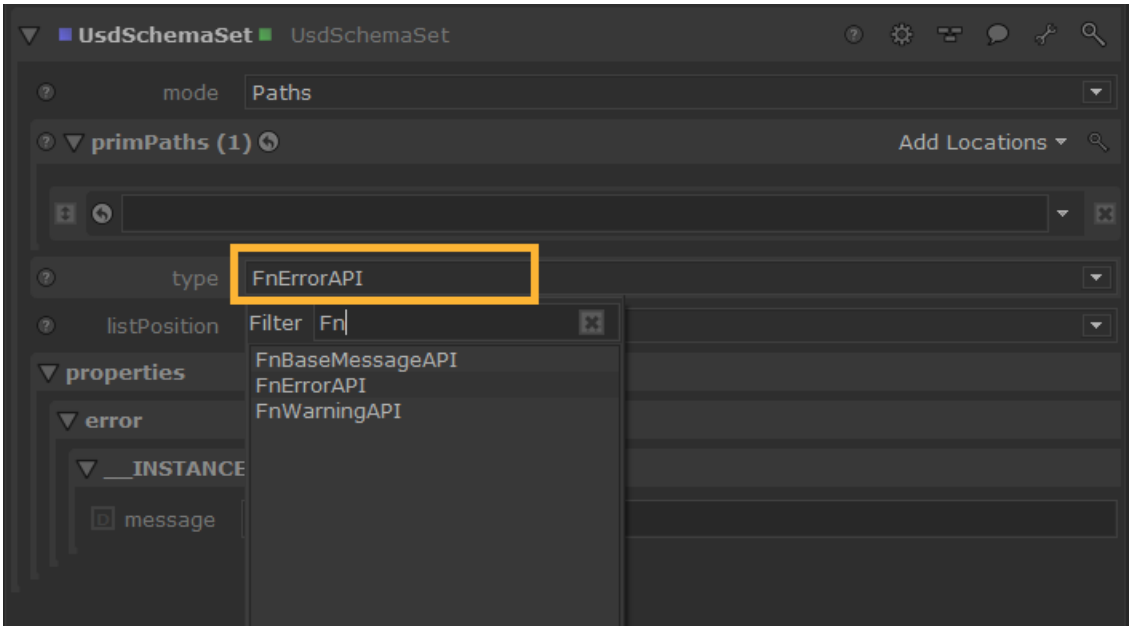
USD Error Handling

We have added new **Error schemas** to help identify issues, for example the **FnErrorAPI** schema and **FnWarningAPI** schema.

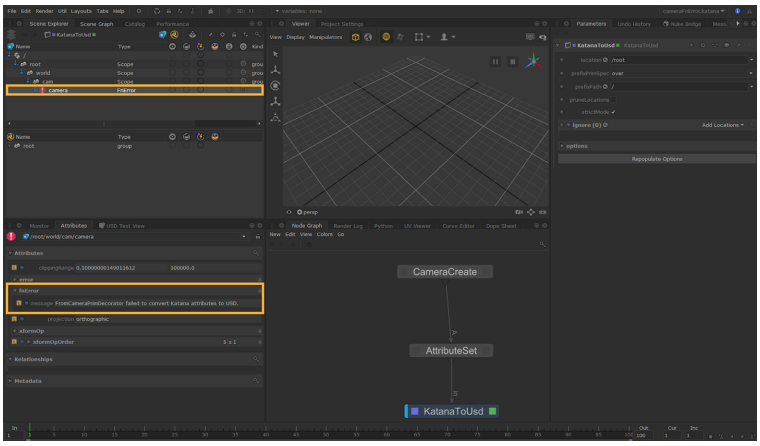
These are used in the KatanaToUsd node for when conversion would be incomplete or in an error state.

The failing prims would have the FnErrorAPI schema or the FnWarningAPI schema applied to them which changes the primType of the failed prim and includes an error log in the USD code.

The schemas can be found in the UsdSchemaSet node and can be used to set a message on a prim:



An example can be found below with more info available in the developer guide.

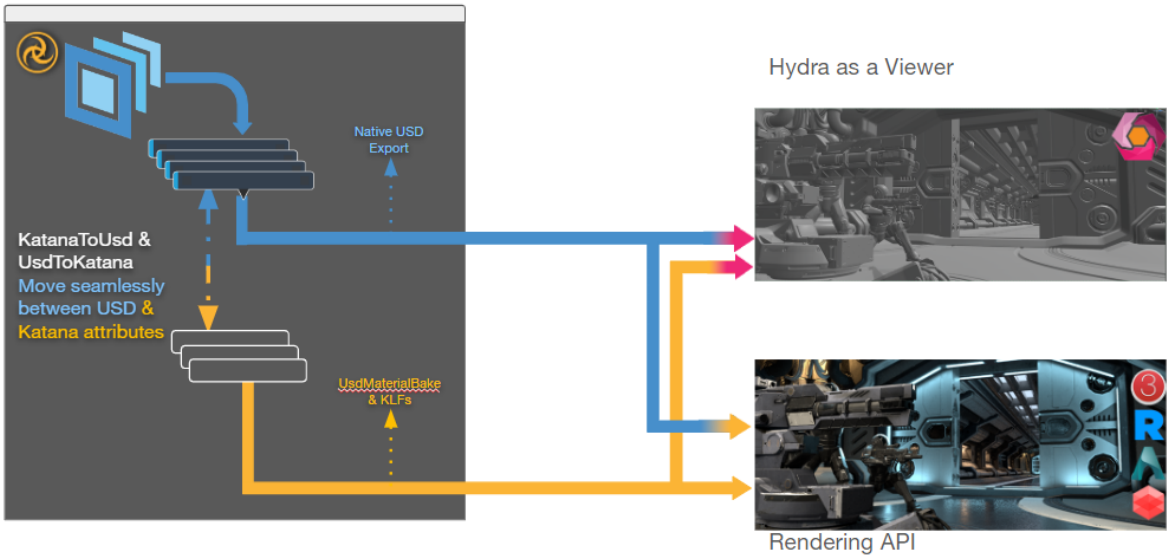


Bridging between Katana Geolib and USD

Tools have been introduced to help you integrate Katana Geolib and USD in your projects in a versatile way. If you are not fully set up for native USD, you may need to make exports and conversations as you go. Although Katana will allow you to work with both Geolib and USD at the same time, and make the required conversions on the way to the Hydra Viewer or the render, these tools bring added flexibility:

- The UsdToKatana node lets you convert USD data into Katana data so that you can continue to work with Katana's Geolib nodes. Additionally, the KatanaToUsd node converts Katana data to USD, allowing you to seamlessly "round-trip". These nodes allow for this conversion to happen directly within a Katana project, without the need to bake them out and read them in. See Conversion Nodes - KatanaToUsd/ UsdToKatana.
- UsdMaterialBake offers a versatile framework for exporting materials and lights.
- Native USD Layer Export enables you to export directly from native USD.

This diagram shows a visual representation of the conversions between USD and Katana Geolib using these tools.



Conversion Nodes - KatanaToUsd/ UsdToKatana

The KatanaToUsd node (and the UsdToKatana node) are conversion nodes used to get data from Katana's Geolib space to USD's native space and vice versa.

KatanaToUsd will look in the **SdrRegistry** for the USD equivalent of the Geolib input data. It will set the USD primitive to an error state if it can't find a suitable type.

This works with a plugin system to support the conversion of different types. This allows for the implementation of **custom primdecorators** to add any additional types to the list of supported convertible types.

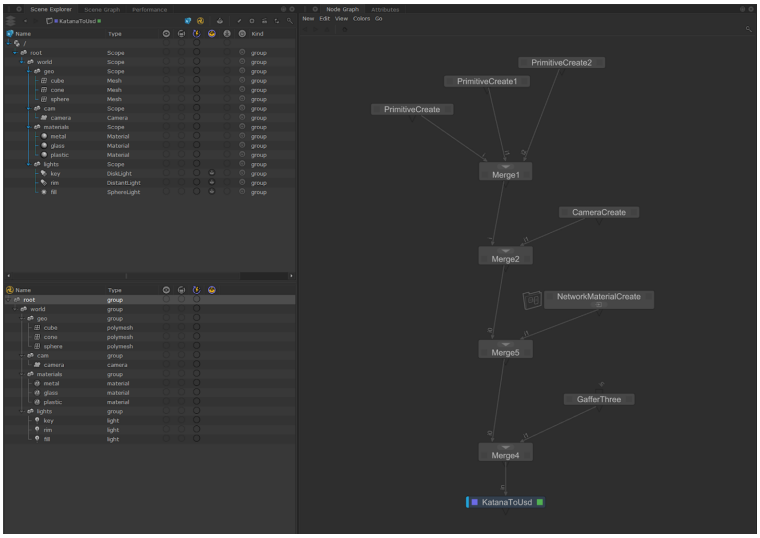
See types of conversions that this node supports

- Groups
- Assemblies
- Components
- Polymeshes
- Subdivided meshes

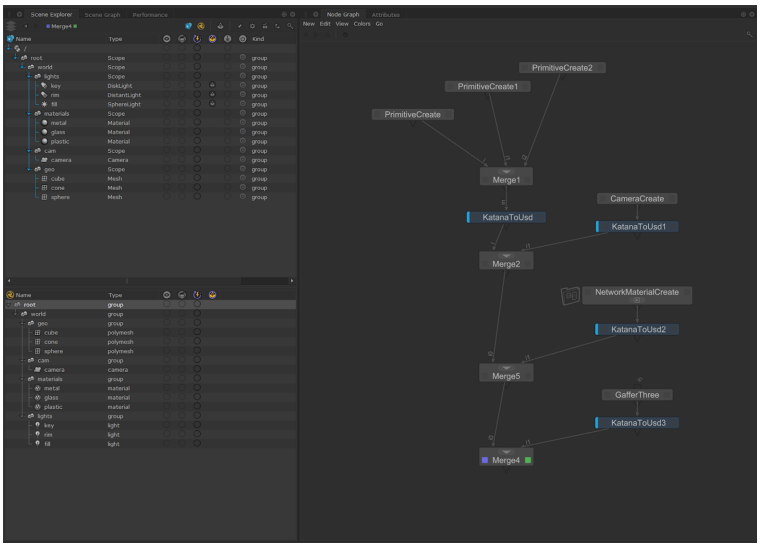
- Cameras
- Materials
- Lights
- Light Linking
- Facesets
- Custom Attributes
- Timesamples
- Visibility
- Volumes
- Custom Primdecorators

Converting data using KatanaToUsd - Example:

In the example below we are using the KatanaToUsd node to convert all of our Geolib data to USD data, including geometry, cameras, materials and lights.

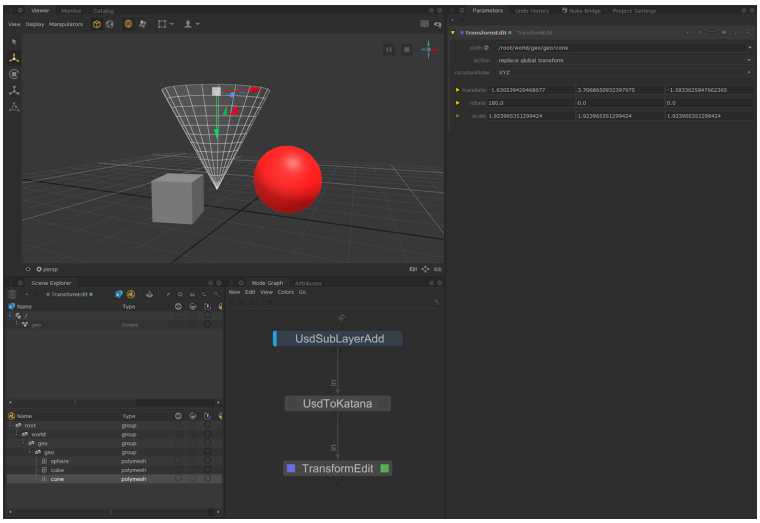


Note: Since the KatanaToUsd node provides a continuous conversion, it uses a caching strategy to improve performance. However, changing an upstream parameter requires the node to reconvert its input. It is recommended to use multiple KatanaToUsd nodes in your project as well as using this in conjunction with the UsdLayerExport node and its '**useExported**' functionality to improve performance.





UsdToKatana converts USD Native data to Geolib data and uses the same underlying infrastructure as the UsdIn node (as the UsdIn node will immediately do this conversion on all incoming USD layers). The data can then be used for Katana-specific workflows.



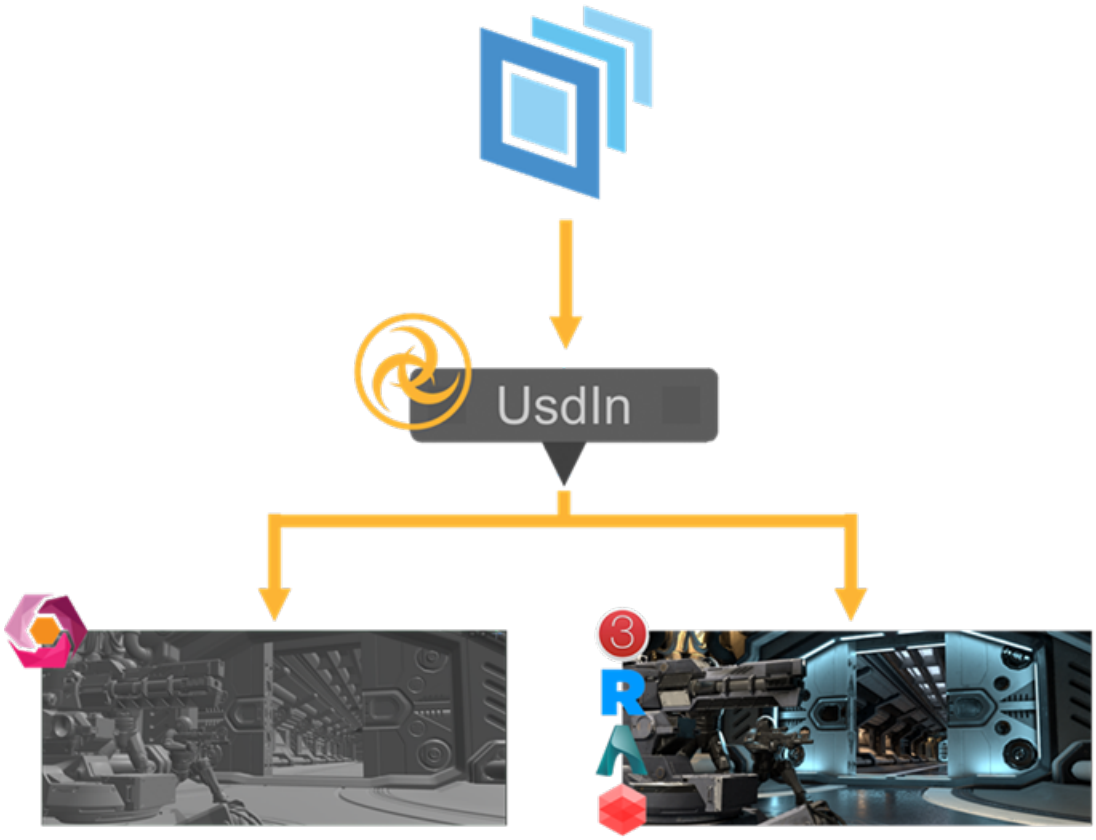
Pre-existing UsdIn Workflows

Before the introduction of native USD nodes and tools, we offered ways to integrate USD into Katana workflows, such as UsdIn, which uses Katana's proprietary data called **geolib**.

These workflows are still possible in Katana and may be helpful to use instead of native USD in certain cases:

- Loading in USD Assets with UsdIn
- Setting up USD Lights
- Setting up UsdPreviewSurface Materials
- Loading USD Plug-ins into Katana

This diagram shows a visual representation of the transfer between between USD and Katana geolib using UsdIn.



Loading in USD Assets with UsdIn

For workflows which require use of USD assets, but are not set up for full native USD, Katana's UsdIn nodes are a useful solution.

The UsdIn node enables you to load a range of USD assets into Katana, including geometry, materials and shading assignments, lighting, and physics.

Essentially, by browsing for a usd file in the Node Parameters, and specifying a Katana scene graph location to load the USD contents, the asset will be imported.

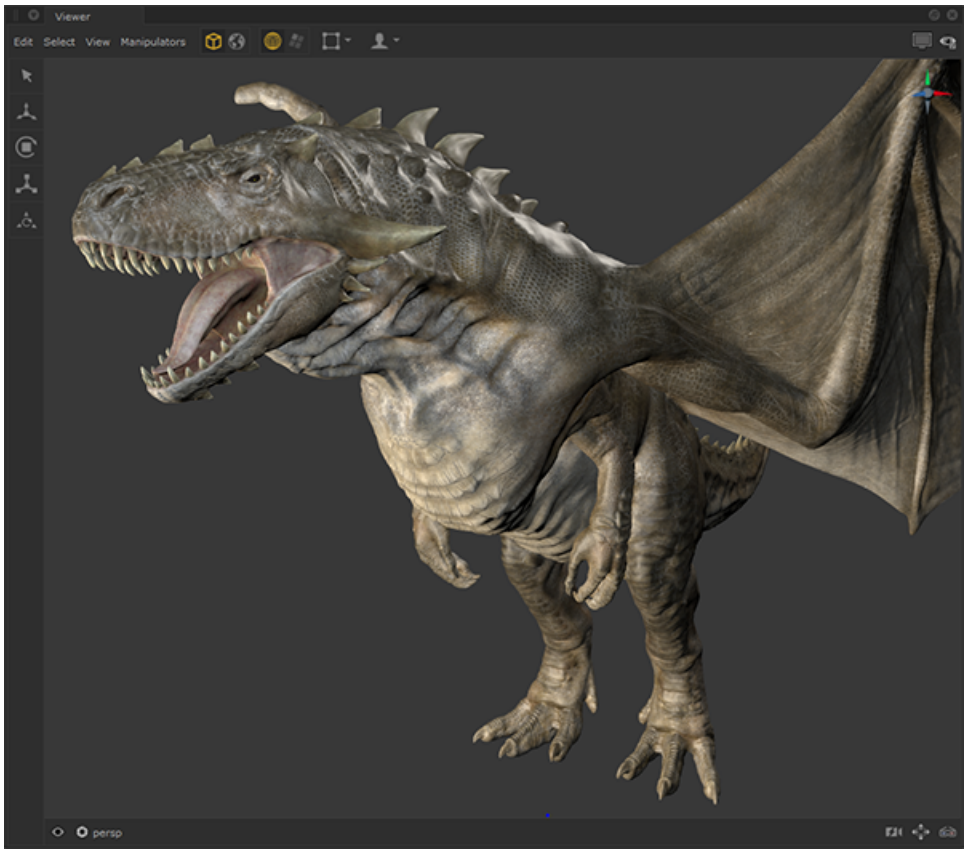
There are several UsdIn nodes for a variety of purposes.

- **UsdInActivationSet**
- **UsdInAttributeSet**
- **UsdInDefaultMotionSamples**
- **UsdInIsolate**
- **UsdInMotionOverrides**
- **UsdInResolveMaterialBindings**
- **UsdInVariantSelect**

In recent releases, Katana has been introducing more USD solutions with the addition of a range of entirely native USD Nodes. These new Native USD solutions can work alongside UsdIn to allow greater flexibility, however depending on which you use they will be stored in Katana as different data types. To learn more, see USD Workflows - Choose your route.

Setting up USD Lights

As well as setting up USD materials, it is also possible to set up USD lights and view them in the Hydra Viewer.

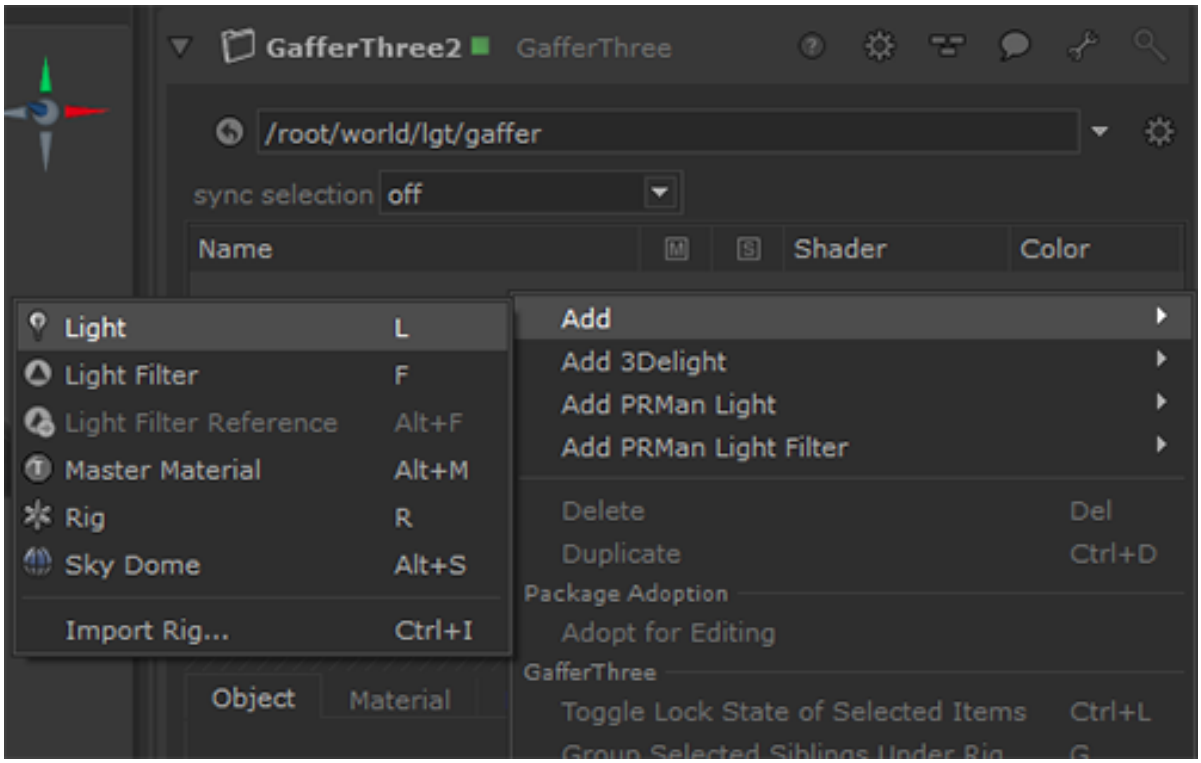


To set up a USD light:

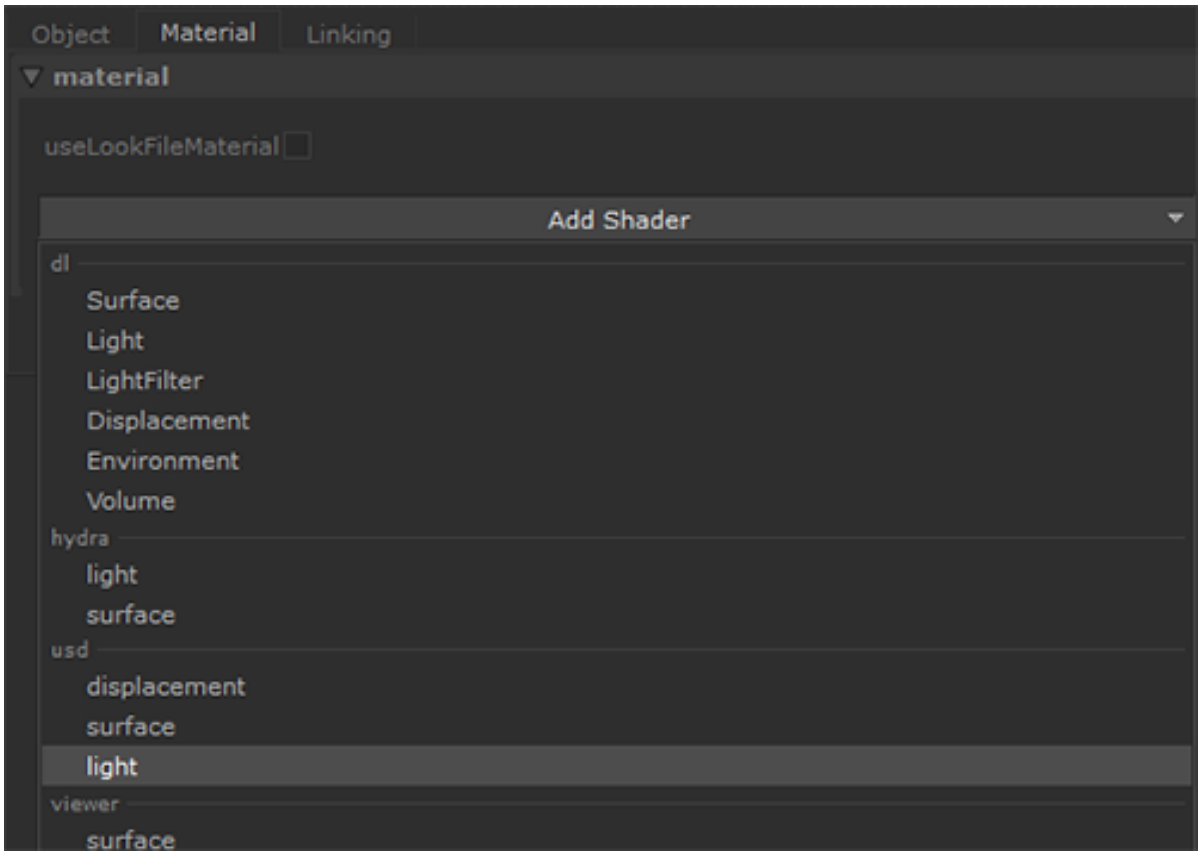
You can use the workflow below to use the **GafferThree** node with a USD Shader.

1. Create a **GafferThree** node and open its **Parameters**, or open the **Parameters** for an existing **GafferThree** node.

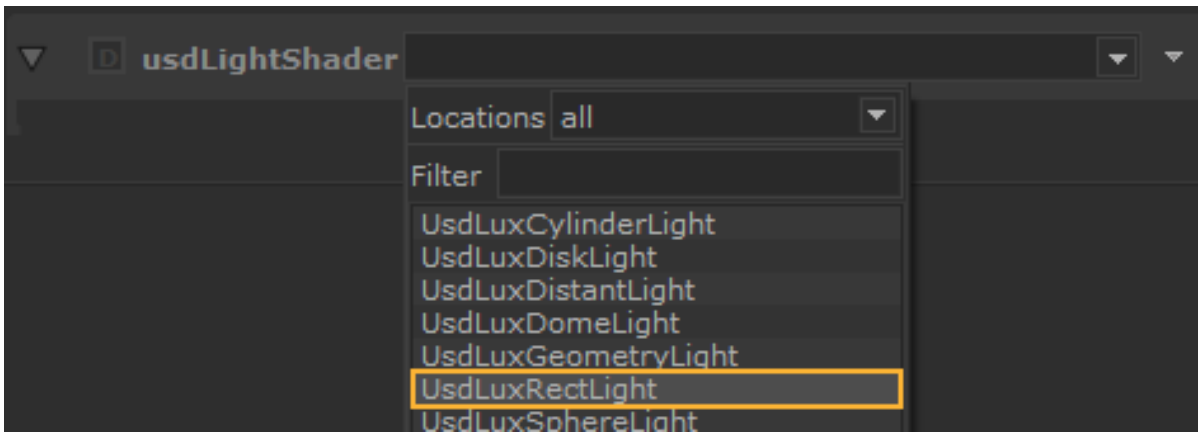
2. In the **GafferThree Parameters** tab, right-click under **Name** and choose **Add** > **Light**.



3. In the **Material** section of the new **light** parameters, click **Add Shader** and choose **usd** > **light**.

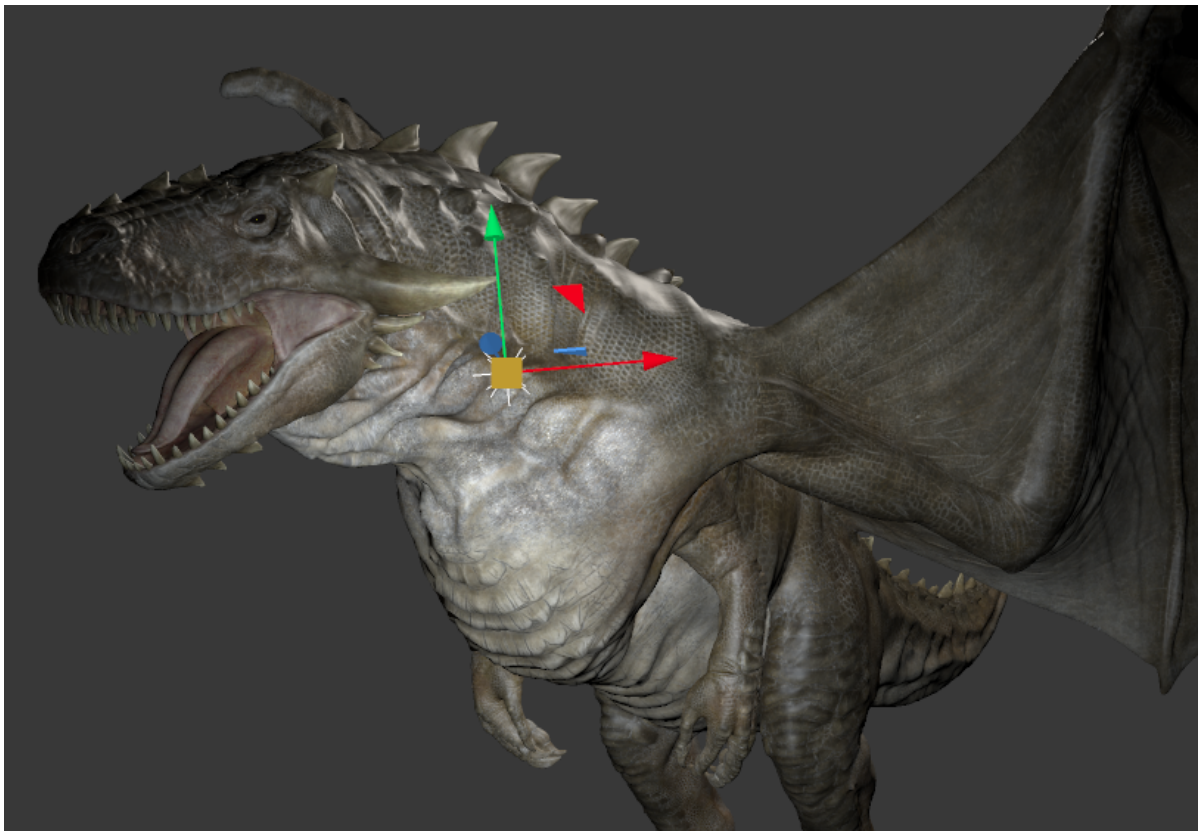


4. In the **usdLightShader** drop-down menu, choose a USD light type, for example, **UsdLuxRectLight**.



The full light material parameters open and you can make adjustments.

5. In the **Viewer** tab, click **View** > **Lighting using USD Lights** to enable the USD lights.
6. Position the light using the **Transform**, **Rotate**, **Scale**, and **Center of Interest** manipulators in the Hydra Viewer.



Setting up UsdPreviewSurface Materials

You can use **UsdPreviewSurface** shading nodes to build USD materials. You can then assign these materials to your object and view the result in the Hydra Viewer.

UsdPreviewSurface shading node types are accessible from the **USD** shading node creation menu inside NetworkMaterialCreate nodes.

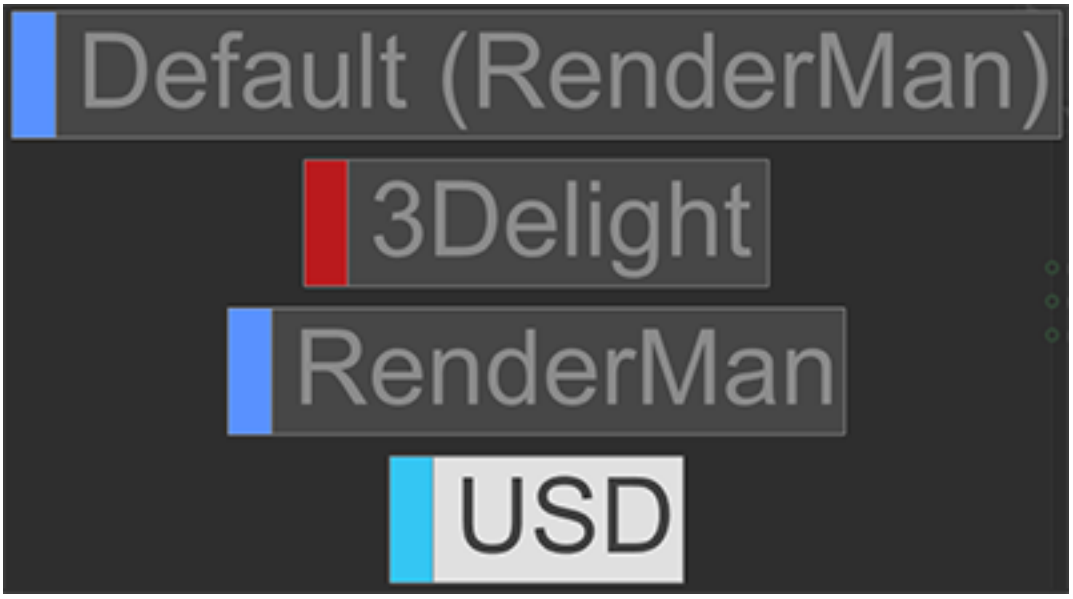
To use **UsdPreviewSurface** to display materials in the Hydra Viewer:

1. Load the USD plug-ins into Katana by following the steps in the Loading USD Plug-ins into Katana topic.
2. Create a **NetworkMaterialCreate** node and jump inside it, or jump inside an existing **NetworkMaterialCreate** node.



Note: If you are using an existing NetworkMaterialCreate node, you may need to refresh the Sidebar Terminal to see the **usd** outputs. To do this, open the **Parameters** for the **NetworkMaterialCreate** node and choose **Shelf Actions** > **Refresh Sidebar Terminal**.

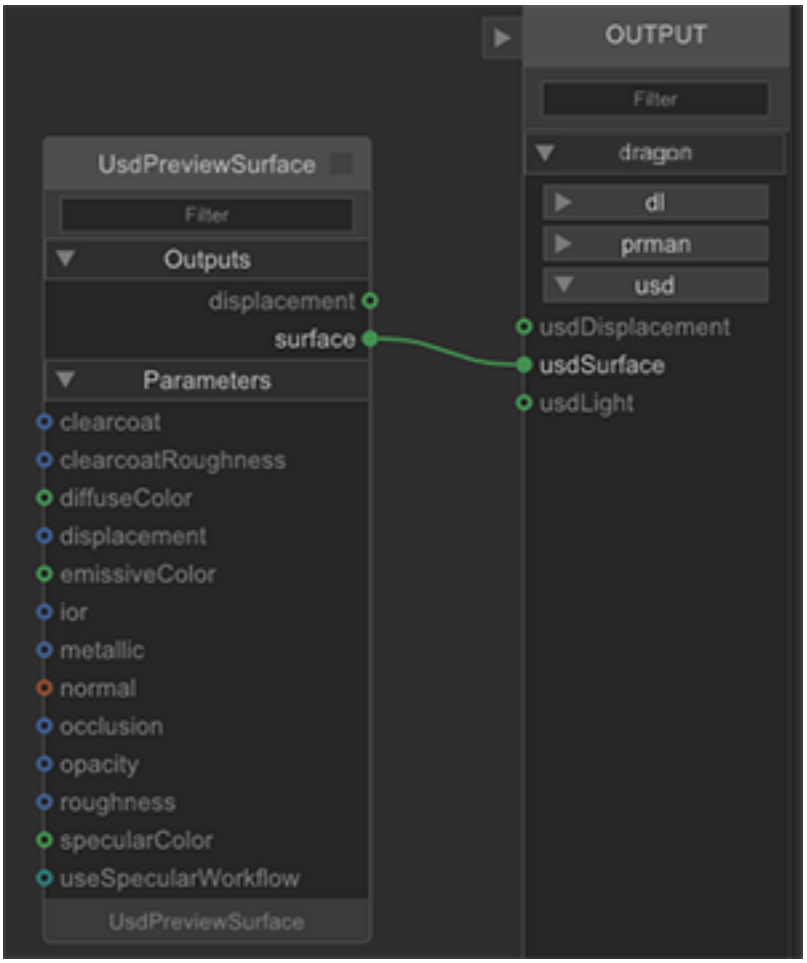
3. Press **Shift** + **Tab** to open the renderer selection menu and choose **USD**.



4. Press **Tab** to open the USD shading node creation menu.
5. Select **UsdPreviewSurface** and place the node.



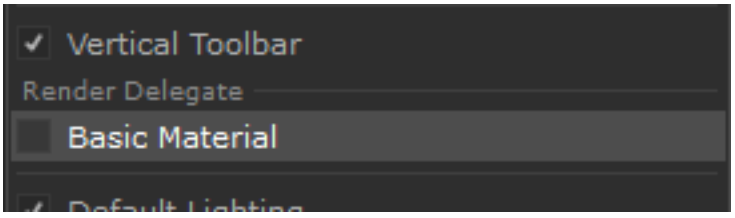
6. Connect the **surface** output from the **UsdPreviewSurface** node to the **usdSurface** input under the **usd** drop-down in the Terminal sidebar.



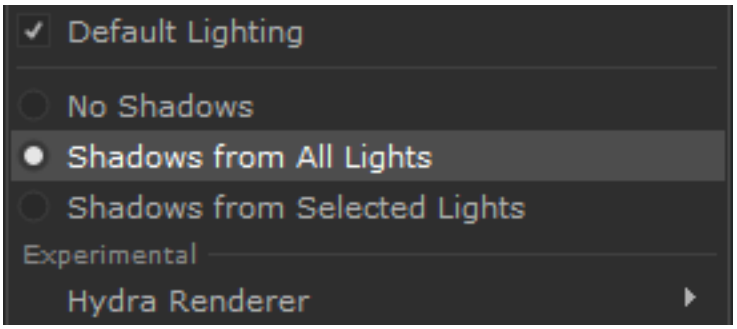
7. Open the **Parameters** for the **UsdPreviewSurface** and make the required adjustments.
8. Use a **MaterialAssign** node to assign the material to your geometry.

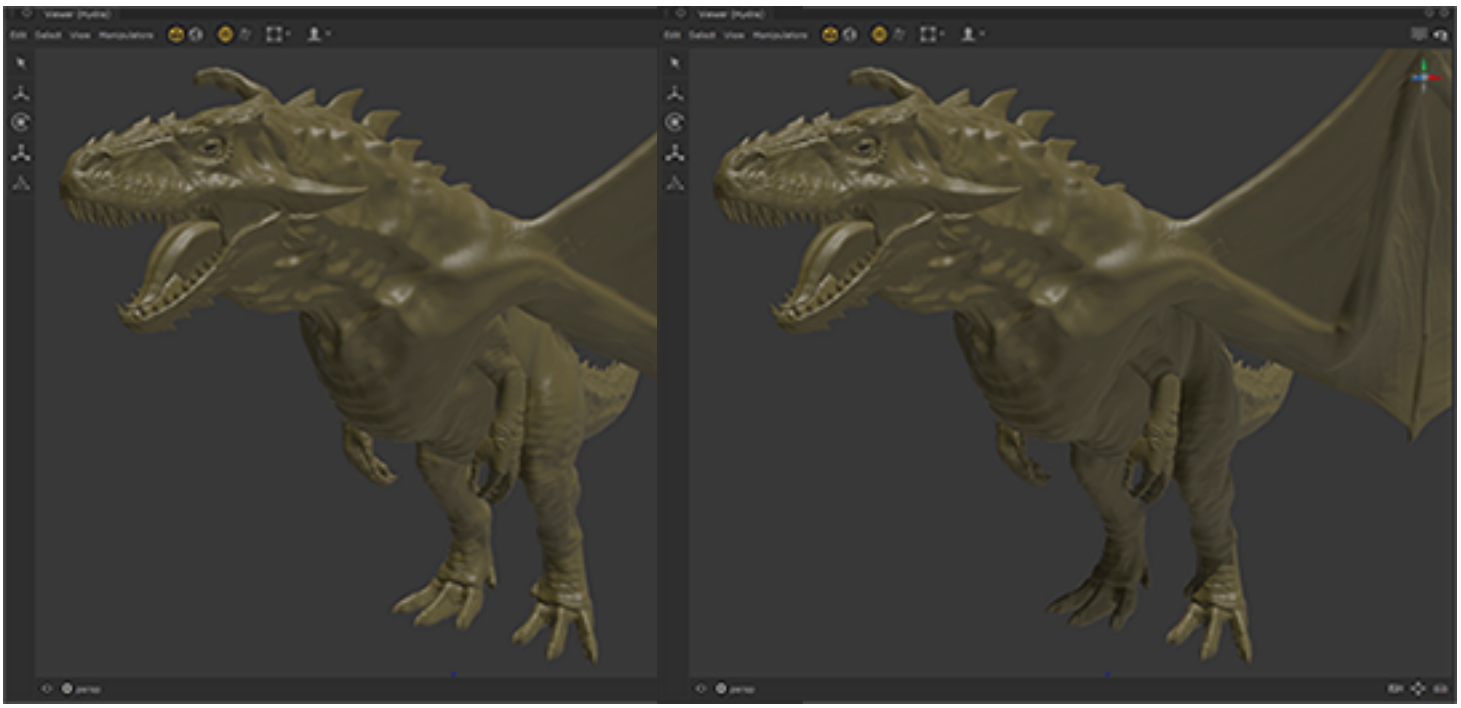
> **Note:** For more information about assigning materials, see the Assigning Materials and Textures section of the Material Basics topic.

9. In the **Viewer** tab, click **View** and disable **Basic Material** to preview your USD material.



In the **Viewer** tab, click **View** and choose **Shadows from All Lights** to preview your material with shadows. If you only want selected lights to cast shadows, you can use the **Shadows from Selected Lights** option.





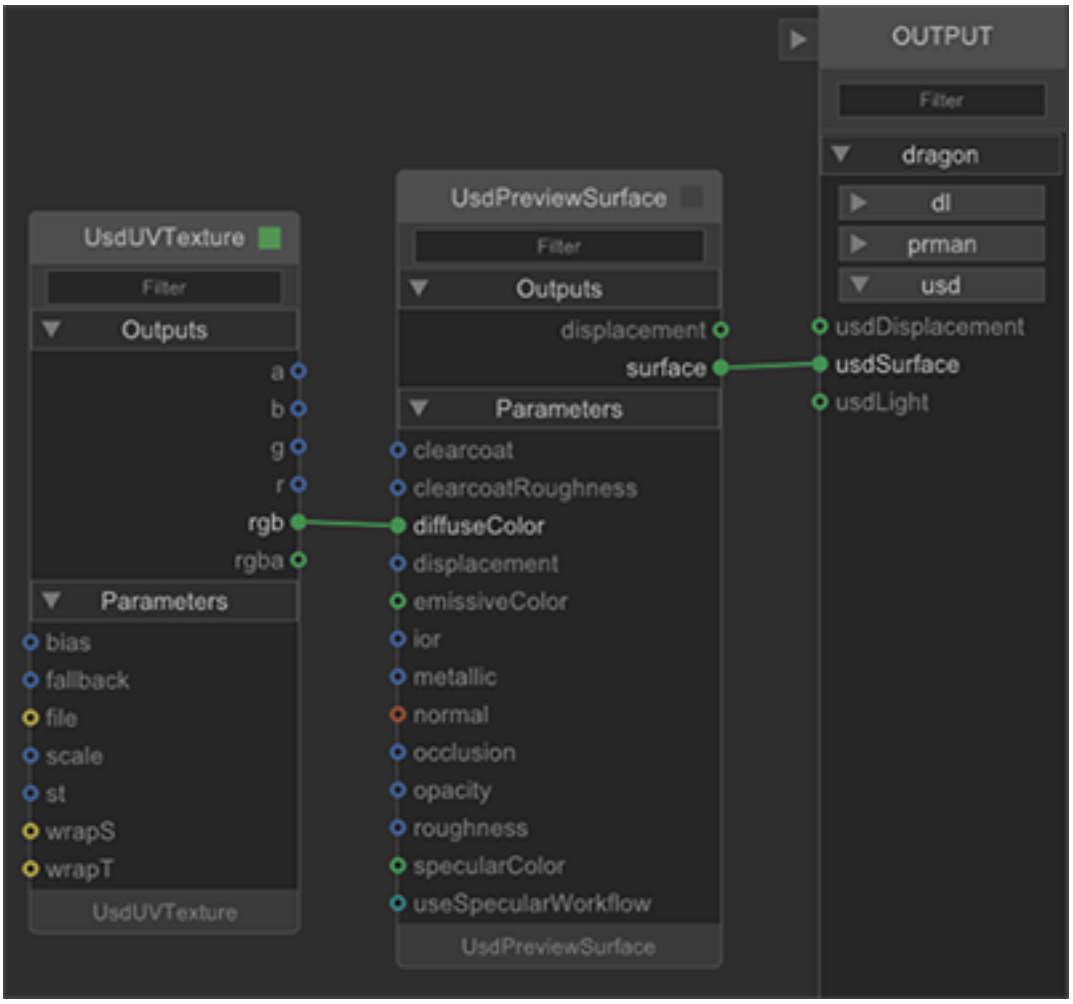
Basic Material off
No Shadows

Basic Material off
Shadows from All Lights

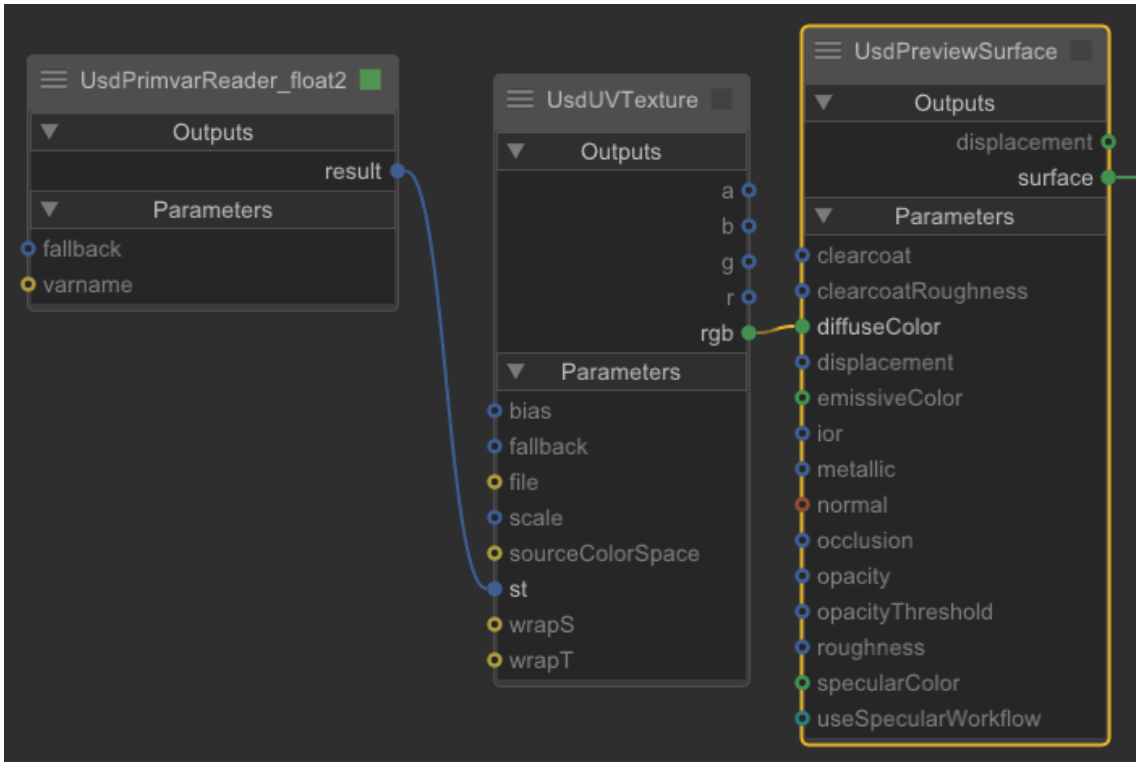
You can assign texture maps to **UsdPreviewSurface** shading nodes using **UsdUVTexture** node types.

To assign a texture map to the **diffuseColor** parameter in your **UsdPreviewSurface** shading node:

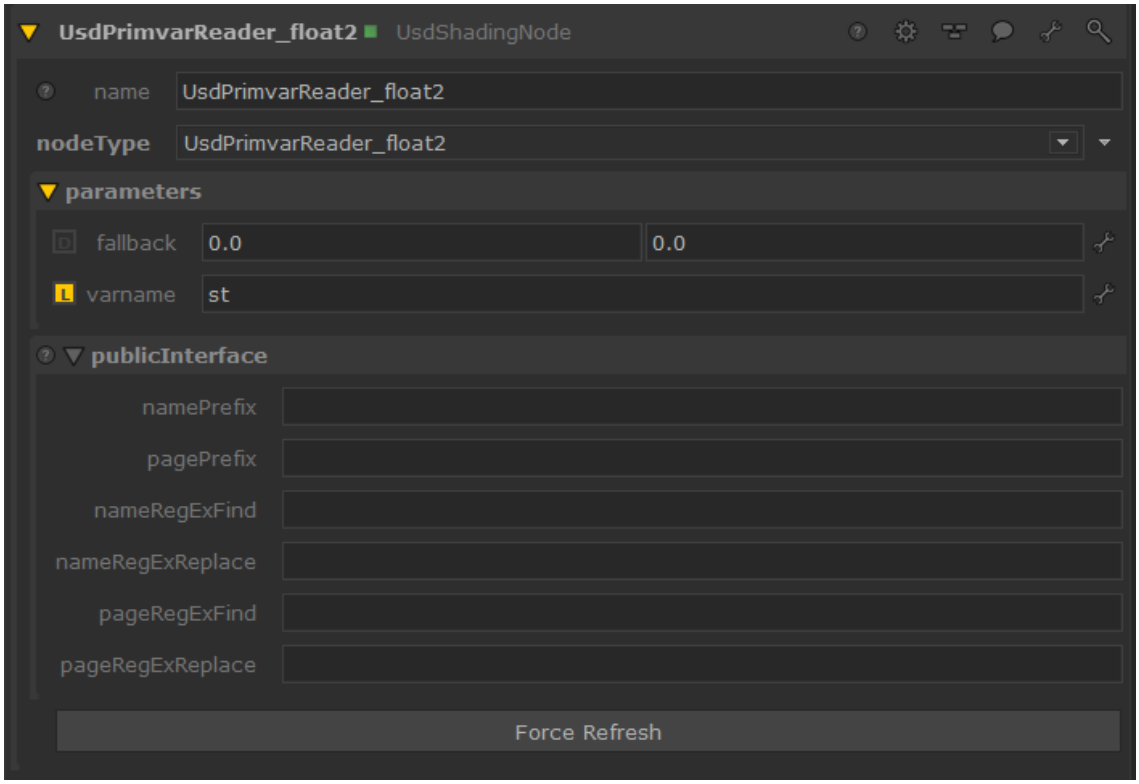
10. Create a **UsdUVTexture** node, open the **Parameters**, and enter the file path in the **file** parameter.
11. Connect the **rgb** output from the **UsdUVTexture** node to the **diffuseColor** input of the **UsdPreviewSurface** node.



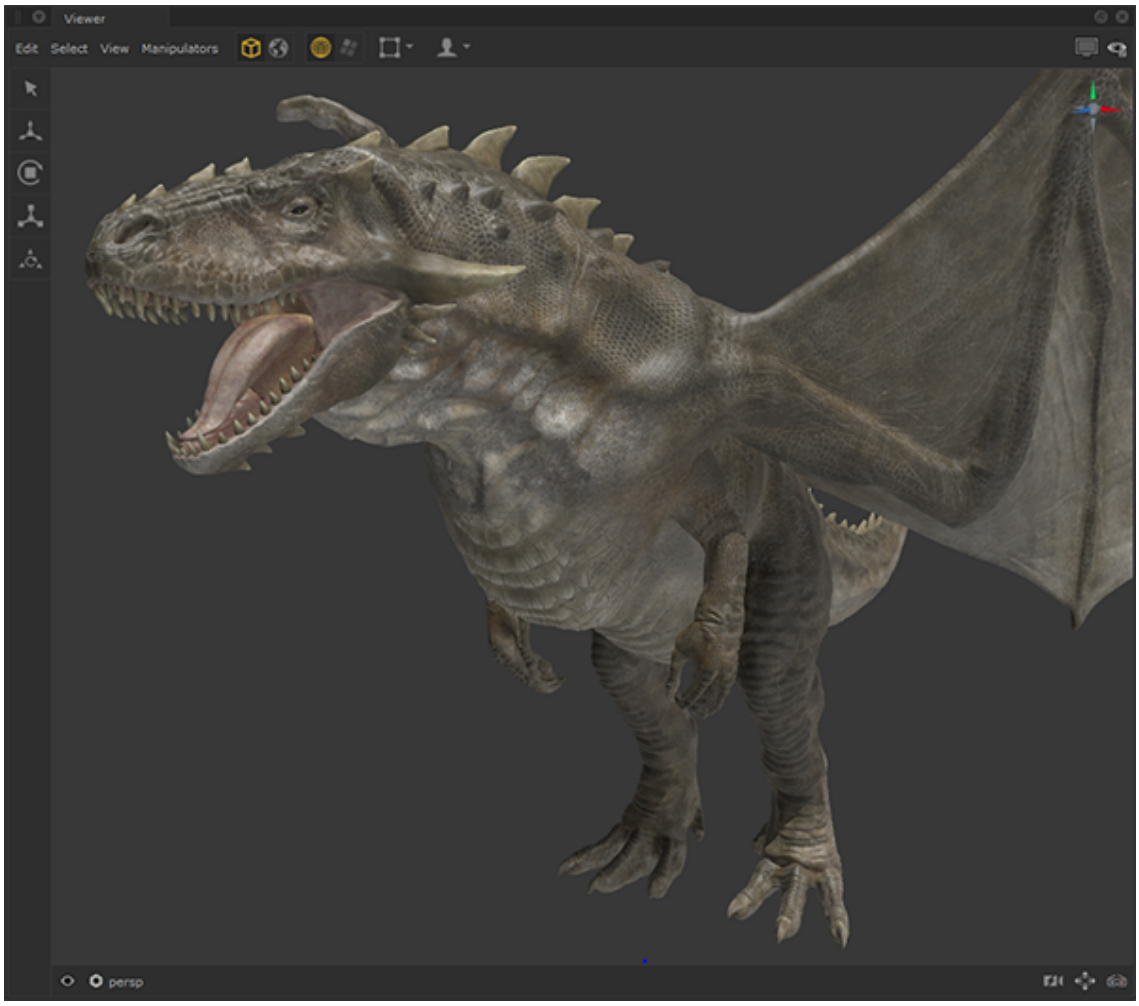
12. Create a **UsdPrimvarReader_float2** node and plug the **result** output into the **st** input on the **UsdUVTexture**.



13. Within the **UsdPrimvarReader_float2**, set the **varname** parameter to **st**.



The texture can now be seen on the object in the Hydra Viewer.



Basic Material off, **Shadows from All Lights**

Video: For a full tutorial series on using USD Preview Surface materials, take a look at our Setting Up USD Preview Surface Materials course.

Loading USD Plug-ins into Katana

In Katana 4.5v1, and later, USD plug-ins are enabled by default. You don't need to define any environment variables, you can use USD plug-ins straight away.

To use the USD nodes inside versions of Katana earlier than 4.5v1, you must first enable the USD plug-ins so that they are loaded when you open Katana.

To do this, you must edit your **KATANA_RESOURCES**, **LD_LIBRARY_PATH** (**PATH** on Windows), and **PYTHONPATH** in your Katana launcher script and add the USD plugin folder.



Note: For more information about creating a launcher script for Katana, refer to these Support articles:
Linux: Creating a Katana Launcher Script for Linux
Windows: Creating a Katana Launcher Script for Windows

1. Add the following lines to your launcher script:

   Note: **<KATANA_ROOT>** represents the path to your Katana install folder, for example: C:\Program Files\Katana4.0v1

   Windows

   ```
   set PATH=%PATH%;<KATANA_ROOT>\plugins\Resources\Usd\lib
   set KATANA_RESOURCES=%KATANA_RESOURCES%;<KATANA_ROOT>\plugins\Resources\Usd\plugin
   set PYTHONPATH=%PYTHONPATH%;<KATANA_ROOT>\plugins\Resources\Usd\lib\python
   ```

   Linux

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<KATANA_ROOT>/plugins/Resources/Usd/lib
   export KATANA_RESOURCES=$KATANA_RESOURCES:<KATANA_ROOT>/plugins/Resources/Usd/plugin
   export PYTHONPATH=$PYTHONPATH:<KATANA_ROOT>/plugins/Resources/Usd/lib/python
   ```

2. Launch Katana using the launcher script and the additional USD node types are available from the node creation menu.

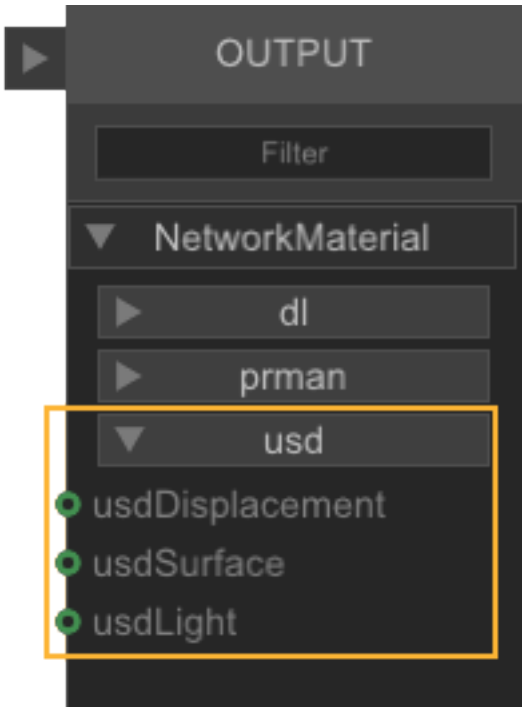


Additional USD nodes



Additional USD shading nodes

A **usd** menu is also loaded on the Terminal sidebar inside NetworkMaterialCreate nodes.

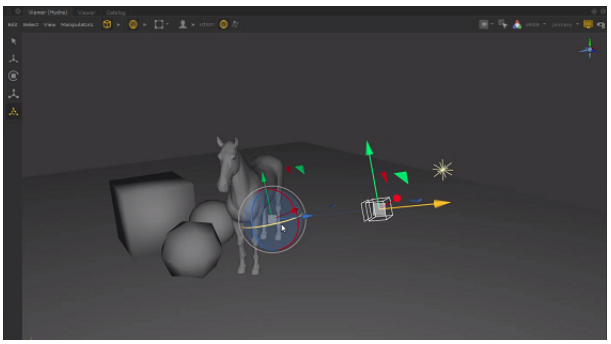


Note: For more information on USD plug-ins for Katana, refer to the Katana USD Plug-ins section in the Developer Guide.

Viewing Your Scene

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.

Katana's **Viewer** is Hydra-powered, providing a fast and flexible way of previewing locations in the scene graph.



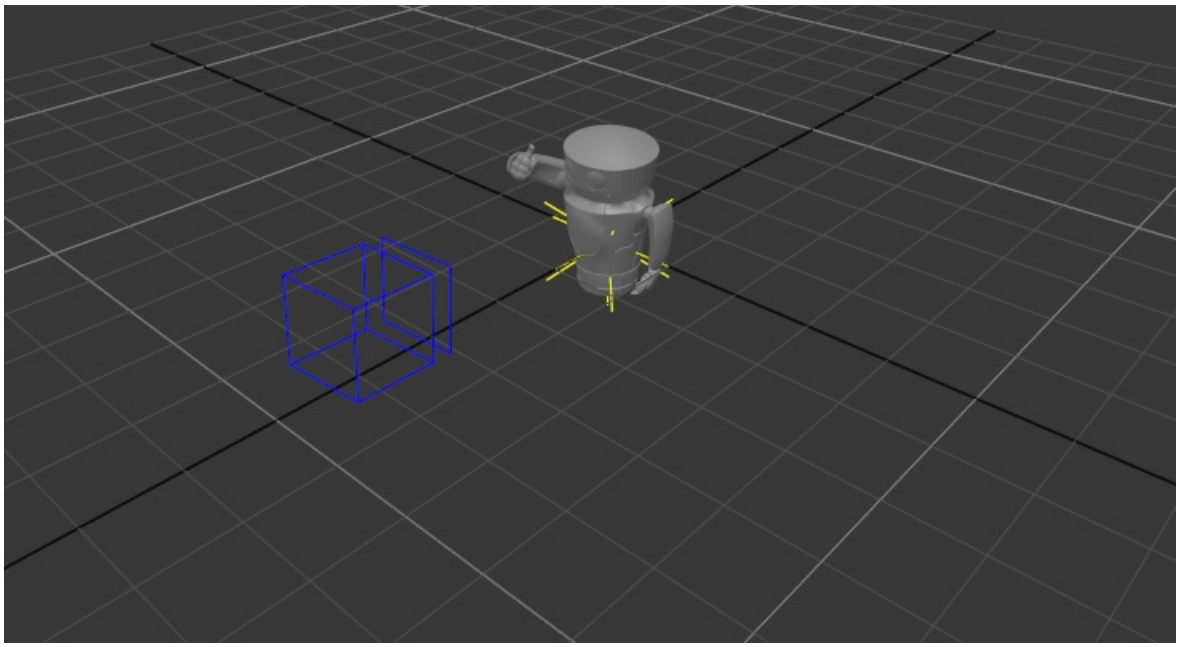
Viewer

Our latest viewer, with massive performance increases and new features.

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.

Katana does not assume a given scene scale. There are no physics solvers, like those in Maya, that need to know what real world measurement a unit represents. If you're using a shader library that has real world units for shade parameters, such as emissive lights with power per unit area, that's handled by the shader implementation rather than Katana.

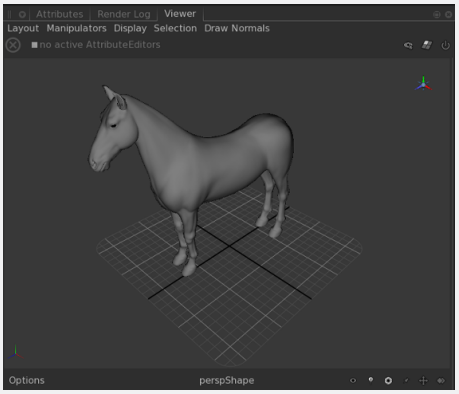
The unit of measure used is up to you, because Katana equates any value as being equal to one unit in 3D space. The image shows a simple scene containing an object and a camera. If the large white square in the Viewer represents one unit of measure, so the smaller squares represent one tenth of that unit. So, if you measured on set in meters, one small square could be equal to a meter, centimeter, or millimeter.

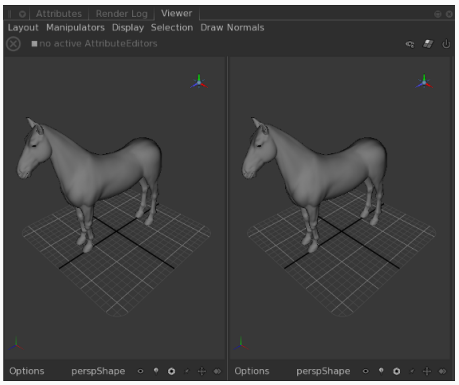
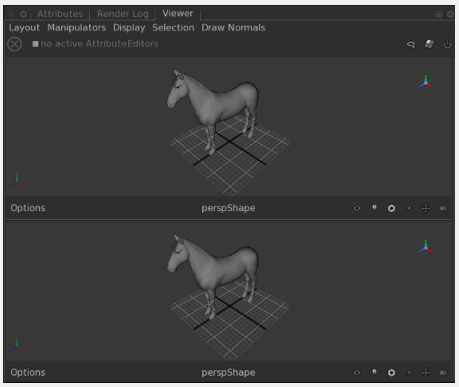
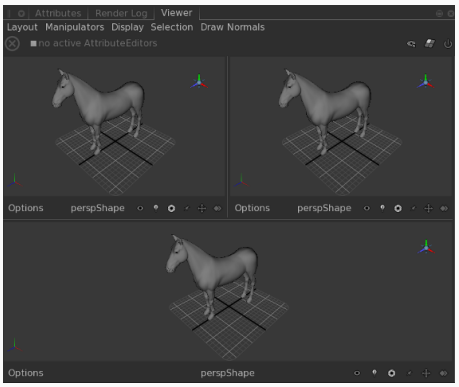


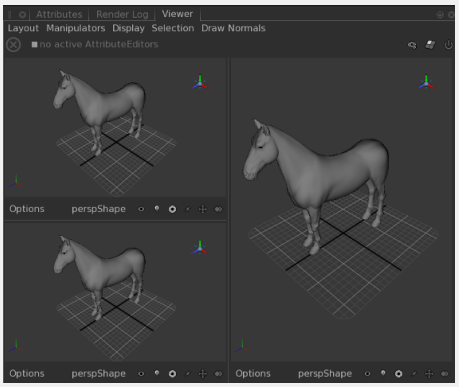
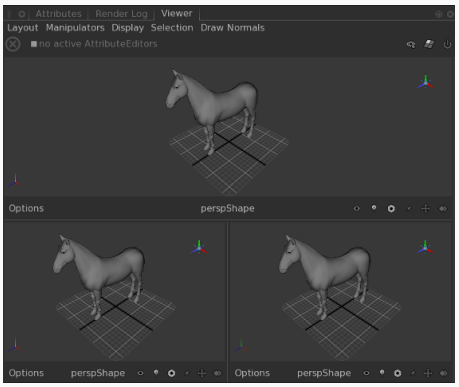
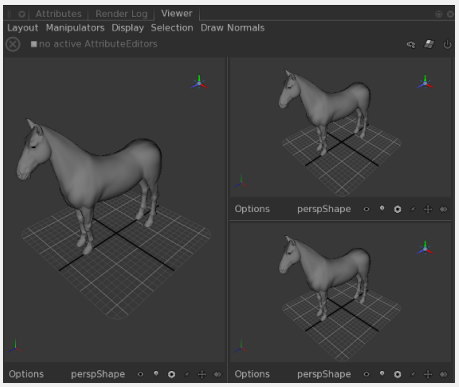
Changing the Layout

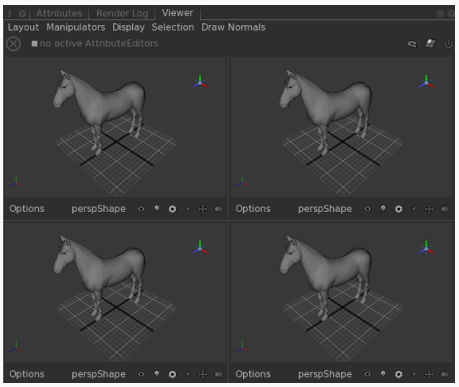
The **Viewer** tab can be split into multiple tabs allowing multiple views of the same scene. Each tab has its own settings for shading, and lighting modes. To split the Viewer tab:

• In the Viewer, select **View** or> **[viewer configuration].**

Viewer Configuration	Example
• **Single Pane** - a single tab takes up the whole **Viewer**. This is the default.	

Viewer Configuration	Example
• **Two Panes Side-by-Side** - this displays two tabs split vertically, sitting side-by-side.	
• **Two Panes Stacked** - this displays two tabs split horizontally, one above the other.	
• **Three Panes Split Top** - this displays three tabs: one large on the bottom, and two more split vertically above.	

Viewer Configuration	Example
• **Three Panes Split Left** - this displays three tabs: one large on the right, and two more split horizontally on the left.	
• **Three Panes Split Bottom** - this displays three tabs: one large on the top, and two more split vertically below.	
• **Three Panes Split Right** - this displays three tabs: one large on the left, and two more split horizontally on the right.	

Viewer Configuration	Example
• **Four Panes** - 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**).

Selecting Within the Viewer

You can use standard selection behavior within viewers.

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.

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.

Using the Viewer

Katana's **Viewer** is Hydra-powered, providing a fast and flexible way of previewing locations in the scene graph.

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.

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.

To access the Viewer, navigate to **Tabs** > **Viewer**.

The **Viewer** incorporates **Lighting Tools**, an enhanced lighting environment for artists. For full information about **Lighting Tools** workflows, see Lighting Tools.

Changing the View Position

You can change which camera or light you are viewing through and their position and orientation. This makes light and camera positioning easy. To change the view's current position and orientation:

Shortcut	Action
Alt+left-click and drag	Tumbles the view around its center of interest.

Shortcut	Action
Alt+middle-click and drag	Pans 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.

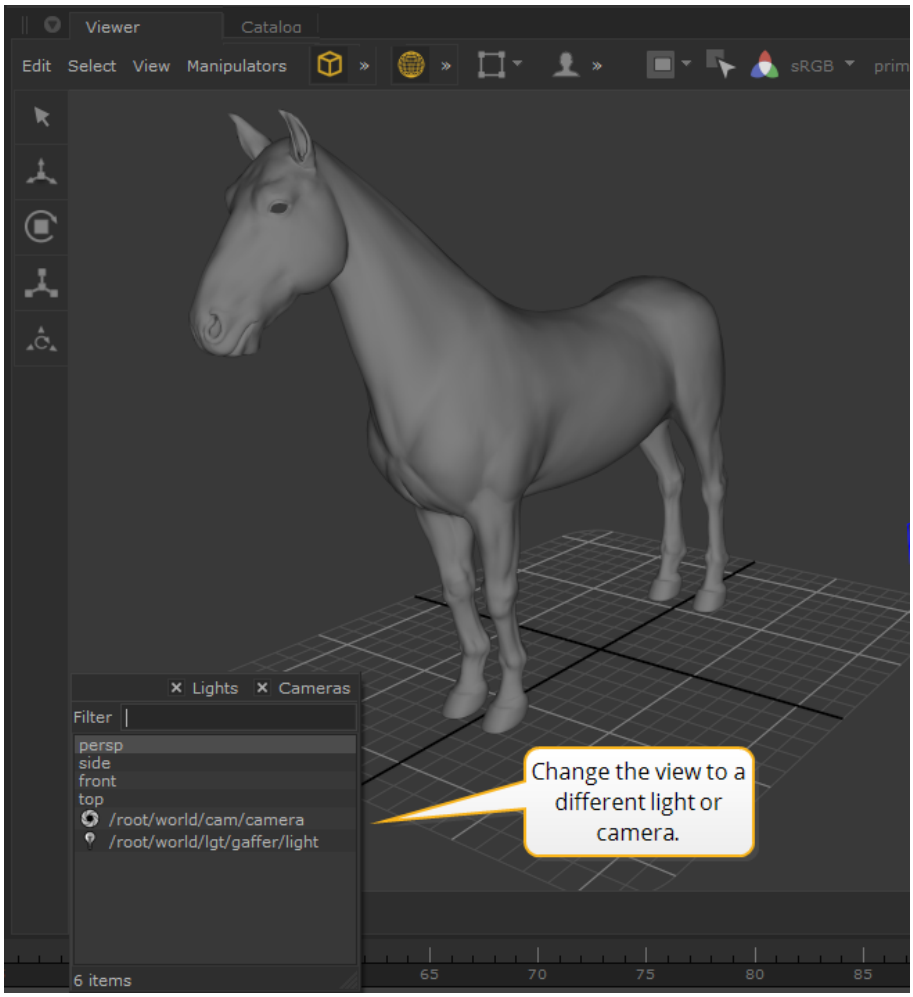
Changing What You Look Through

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. To set this:

1. Click the text at the bottom of the **Viewer** tab (such as **persp**).
   This brings up a list of available lights and cameras.

Note: The default cameras are **persp**, **side**, **front**, and **top**.

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.
     Cameras with a scene graph location can be identified by the icon in the filter list.
   - Uncheck the **Lights** checkbox to remove lights from the list.
     Lights with a scene graph location can be identified by the icon in the filter list.
   - Type text into the **Filter** field to only display items that contain the text.
     Text entered into the **Filter** field of the view selection dialogs may contain some basic regular expression patterns, such as ranges [a-z].

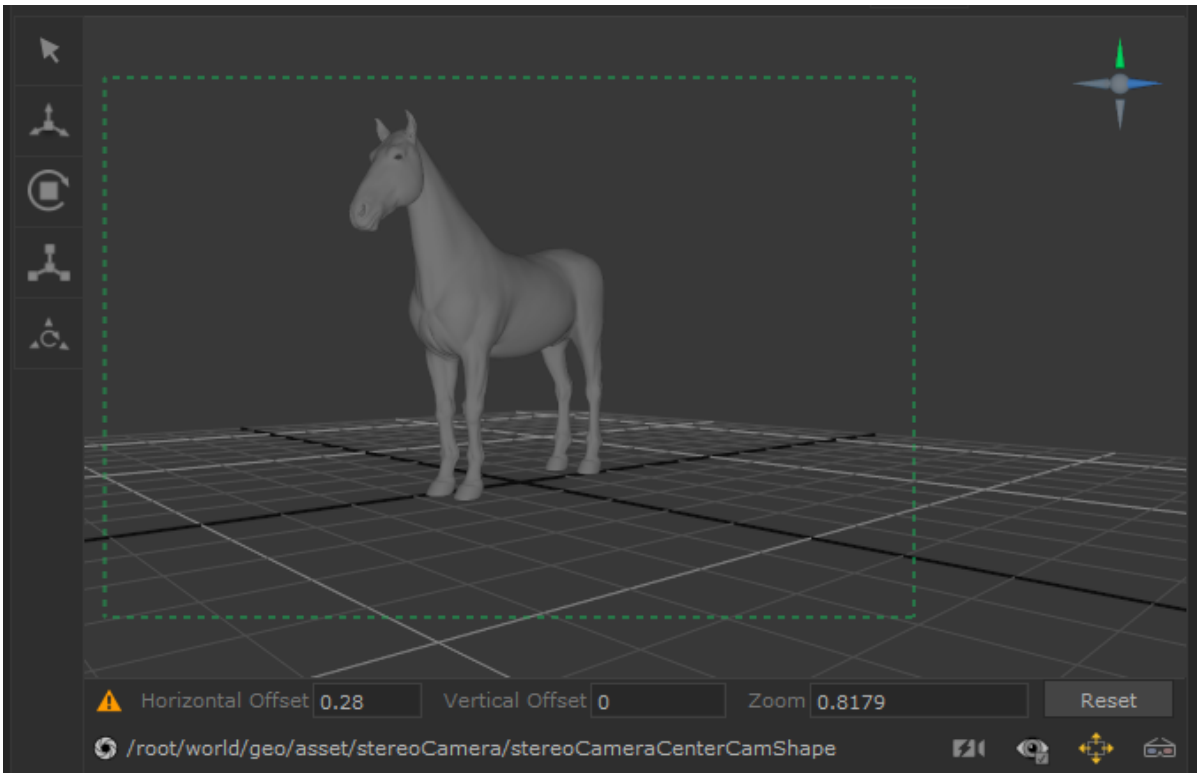


3. Select the required light or camera from the list.

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**, **side**, **front** and **top**).
- The current render camera (such as set with the **RenderSettings** node).

Pan and Zoom

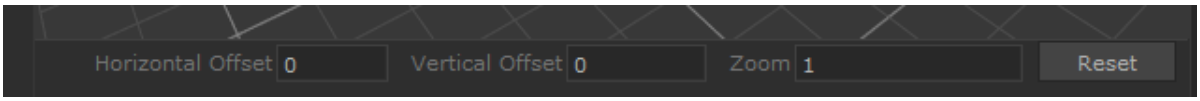
The **Horizontal/Vertical Offset** (pan) and **Zoom** controls allow you to adjust the Viewport position without actually moving the camera. This is useful when a camera has been loaded from another package (such as when representing a camera track) and you don't want to change its position or orientation.



Note: The render window bounding box (the view from the render camera) is shown in the viewport using a green dotted box.

To pan and zoom the selected camera in the Hydra Viewer:

1. Click  to bring up the pan/zoom controls.



2. To make changes to the current view:
   - enter a parameter value to jump to that position, or
   - click-and-hold on the parameter label and slide (scrub) the mouse left or right.

Parameters include:

- **Horizontal Offset** - pan left (negative value) or right (positive value).
- **Vertical Offset** - pan up (positive value) or down (negative value).
- **Zoom** - Zoom in (1 and above) or out (0 to 1).

Tip: All three text fields can be scrubbed by dragging on their names.

3. Click **Reset** to restore defaults.

If offset and zoom values are non-zero, Katana displays a warning icon on the left of the toolbar to indicate that a transform is active.

Selecting Faces

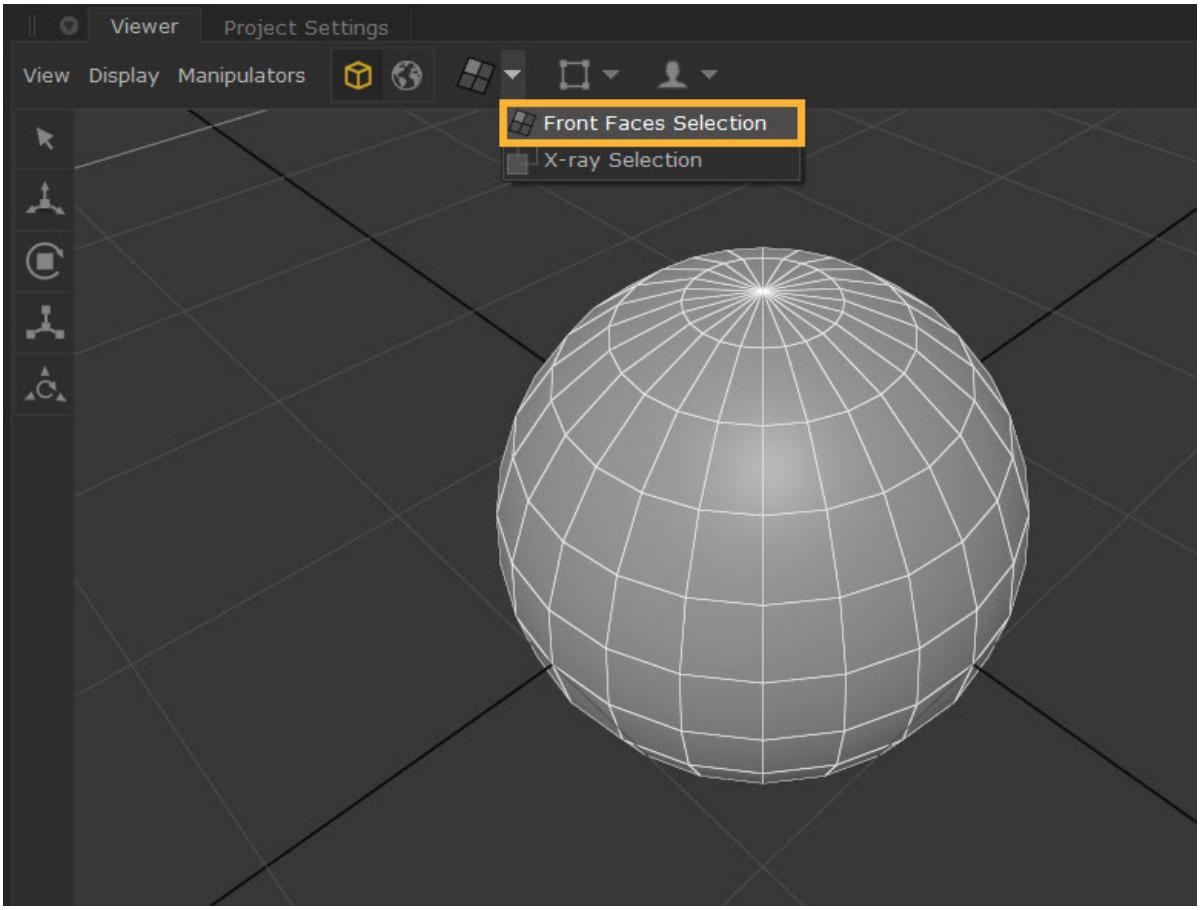
There are two face selection modes - **Front Faces Selection** and **X-ray Selection**:

- **Front Faces Selection** mode selects just the **front** faces within the marquee.
- **X-ray Selection** mode selects **all** the faces within the marquee including front and back faces.

Selecting Front Faces

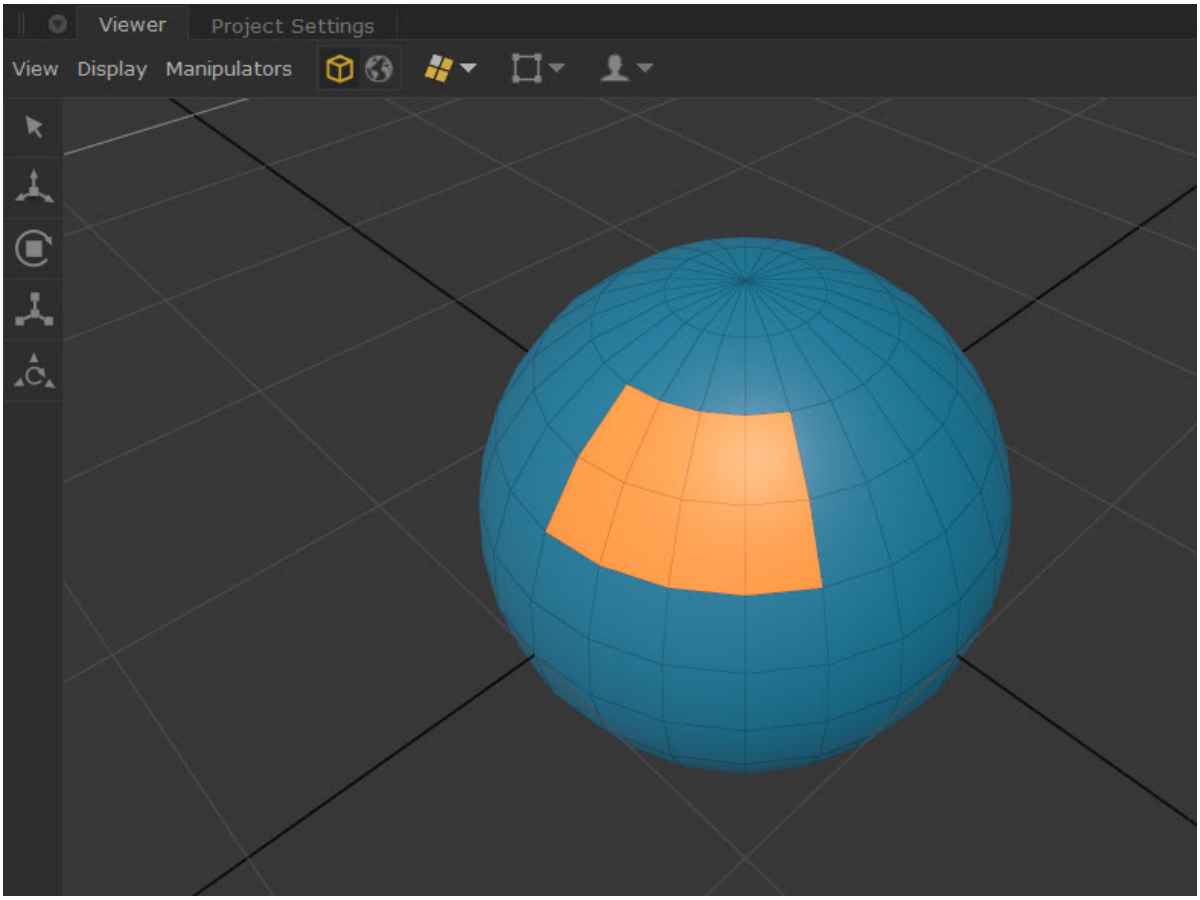
Front Faces Selection mode selects just the **front** faces within the marquee. To use this mode:

1. In the Viewport, click an object.
2. Click the arrow next to the Select Faces button: 
   Select **Front Faces Selection**.



3. Your object displays in a blue color showing that the Front Faces Selection mode is enabled. Select faces on your object using the marquee tool.

   Your object displays the selected faces in an orange color.

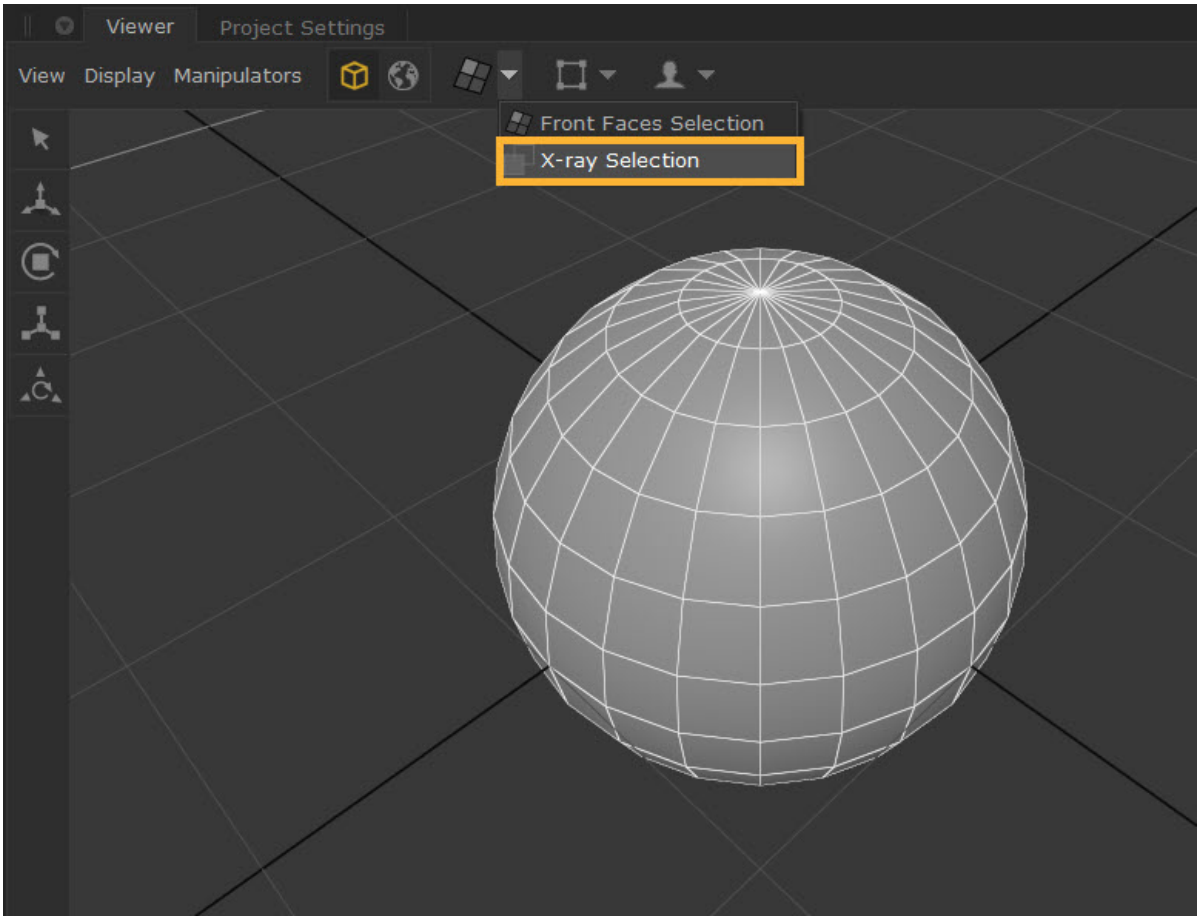


4. Hold **Ctrl** + **Shift** to select additional faces.
5. Hold **Shift** and then click and drag on selected faces to invert the selection.

Selecting Faces via X-ray Selection

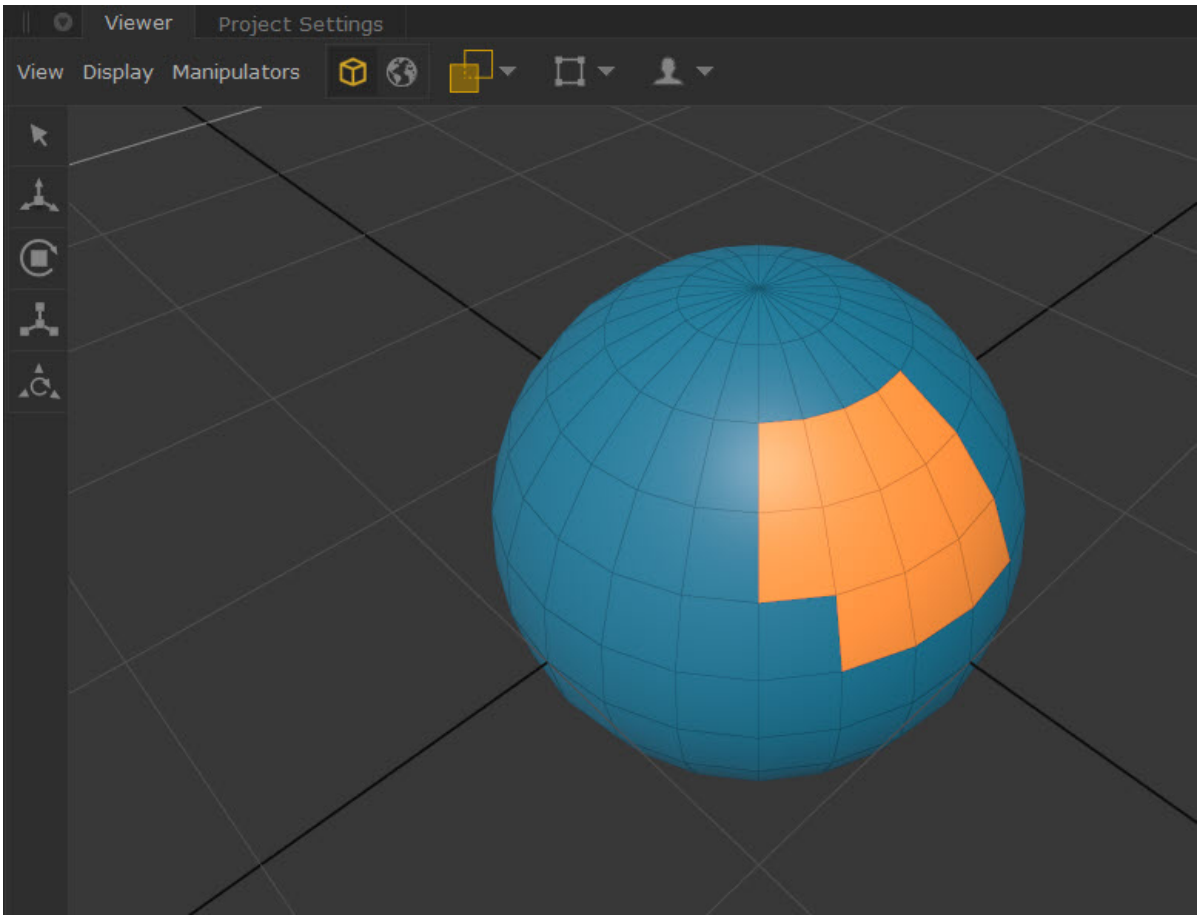
X-ray Selection mode selects **all** the faces within the marquee including front and back faces. To use this mode:

1. In the Viewport, click an object.
2. Click the arrow next to the Select Faces button: Select **X-ray Selection**.



3. Your object displays in a blue color showing that the X-ray Selection mode is enabled. Select faces on your object using the marquee tool.

   Your object displays the selected faces in an orange color.



4. Hold **Ctrl** + **Shift** to select additional faces.
5. Hold **Shift** and then click and drag on selected faces to invert the selection.

Deselecting Faces

To deselect faces you can either:

• Deselect some faces by holding **Ctrl** and dragging your cursor over the selected faces.

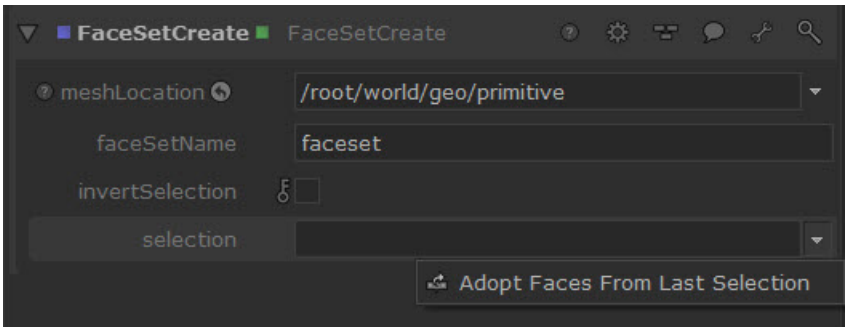
OR

• Click in the Viewport to undo your face selection all together.

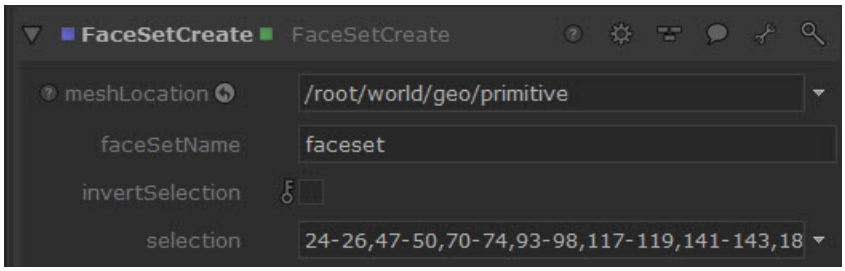
Storing Face Set Selections

You can also use the FaceSetCreate node to store face set selections.

1. Add a FaceSetCreate node to your node graph.
2. Add the location of the mesh under **meshLocation**.
3. Under **selection**, click the downwards arrow, and **Adopt Faces from last Selection**.



4. This will then store your previous selection within the face set location. You'll see its numerical values appearing in the **selection** field. This can then be used to assign a separate material to a different part of the mesh, if they're not already separated out as individual meshes, for example.



Using Manipulators in the Hydra Viewer

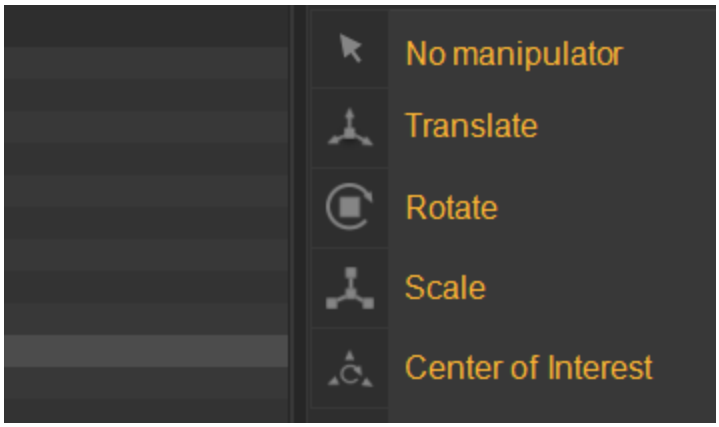
Katana's Hydra Viewer lets you change the size and position of objects and viewpoints, as defined in the scene graph.

A manipulator is only available for locations (created by nodes) that have compatible parameters. For example, if you select a location in the scene graph, you can only move it with a translation manipulator if the nodes that create it have parameters that map to an interactive transform, including:

- a light created by a **GafferThree** or **LightCreate** node,
- a primitive created by a **PrimitiveCreate** node, and
- a mesh with a **Transform3D** node targeted to its scene graph location.

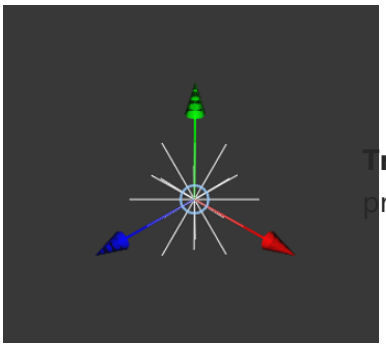
Activating Manipulators

You can activate a manipulator using the **Manipulators** menu, keyboard shortcuts, or the button panel on the left of the **Viewer (Hydra)** tab.

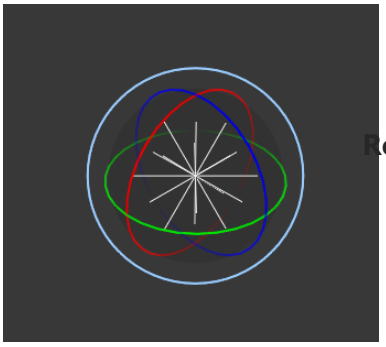


Click and drag the manipulator to use it.

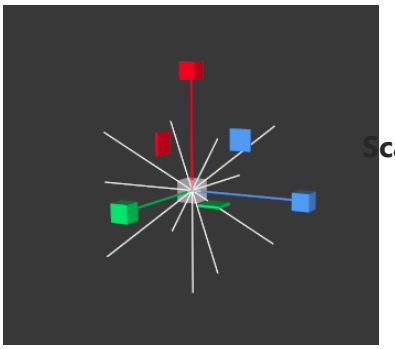
Translate, Rotate, Scale



Translate - click the translate button or select **Manipulators > Translate** or press **W**



Rotate - click the rotate button or select **Manipulators > Rotate** or press **E**

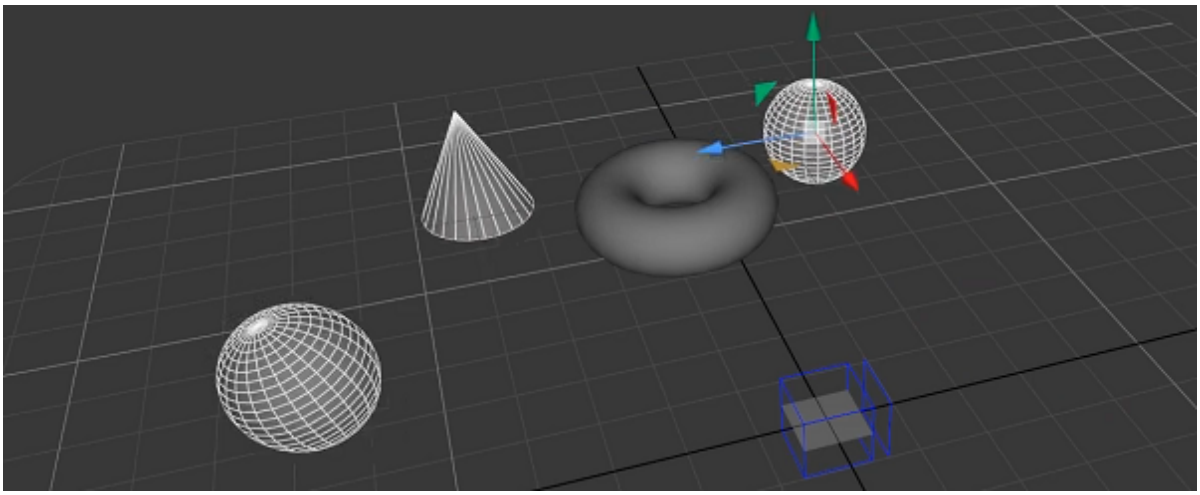


Scale - click the scale button or select **Manipulators > Scale** or press **R**

To deactivate a manipulator, select **Manipulators > No Manipulator** or press **Q**.

Transforming Multiple Objects

The Hydra Viewer allows you to move multiple objects simultaneously. Simply select multiple objects and transform one of the objects. The other selected objects will follow the movement.



The Center of Interest (COI)

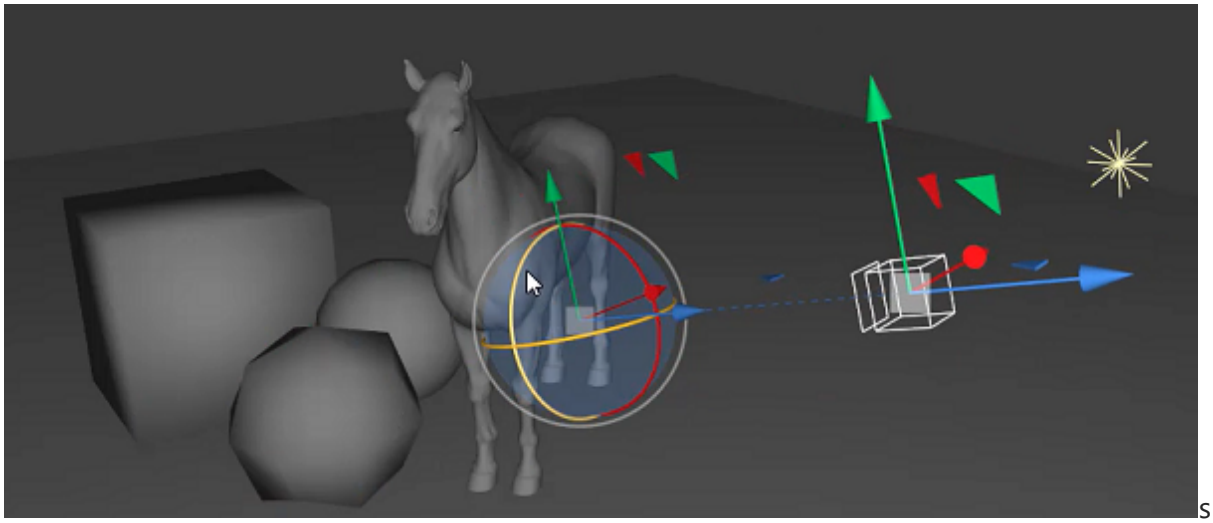
The Center of Interest provides an origin around which to transform a light or camera. This makes it easy to direct a camera or light towards a particular object or region by moving the COI.

Video: Watch this video to learn more about using the Center of Interest.

Activate the Center of Interest by pressing the Center of Interest button, using **Manipulators > Center of Interest**, or pressing **T**.

Transforming a Single Object Around the COI

1. Select an object in the Viewer or its location in the scene graph.
2. Click on the COI button or press **T**.



s

3. Translate or rotate the COI by interacting with the manipulators. The selected object will move relative to the position of the COI.

Transforming Multiple Objects

You can select multiple objects to move using a single COI. When you move the COI each object's own COI will be moved relative to the movements of the COI that you are controlling. When selecting multiple objects, the last objected selected will show its COI manipulator.

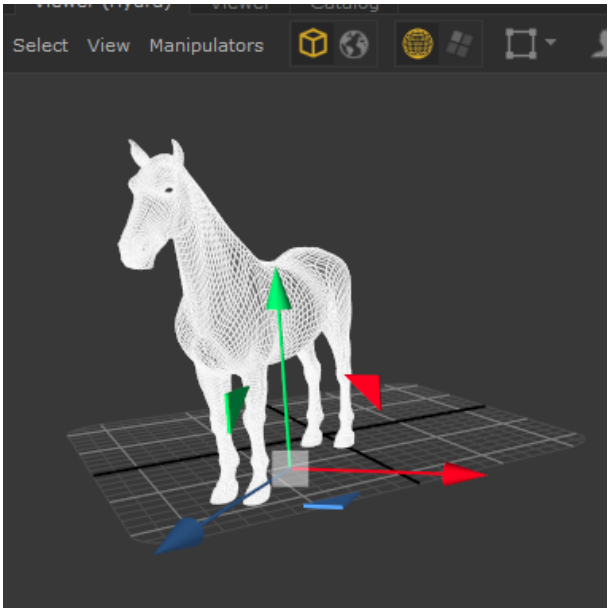
1. In the **Scene Graph**, **Ctrl** and click the objects that you wish to transform. The last object clicked is the one that shows its COI manipulator.
2. Manipulate the COI as needed. The objects will follow.

Object and World Space

You can translate or rotate an object in either object space or world space. Object space is described in relation to the local coordinate system of the object. World space is fixed, common to all objects, and described by the co-ordinate system around the origin of the world in which the objects are located.

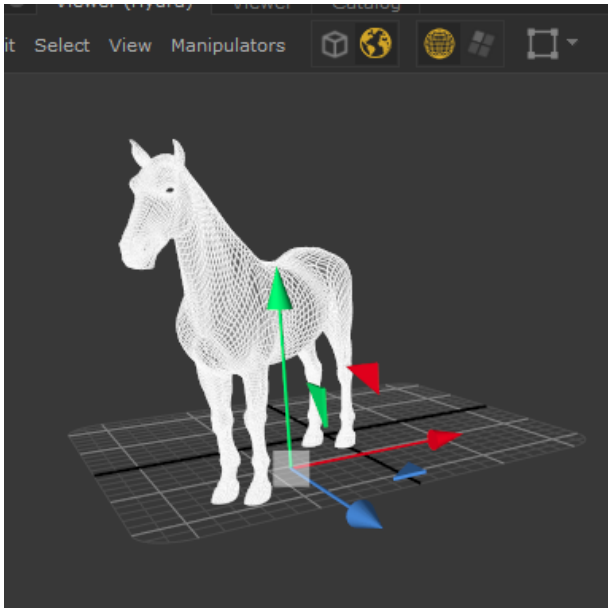
Video: Watch this video to learn more about using Object and World space in Katana's Hydra viewer.

In the Hydra Viewer you can toggle between the spaces using the buttons at the top of the viewer or using the **S** key.



Object Space

Translation and rotation occurs relative to the local coordinate system.



World Space

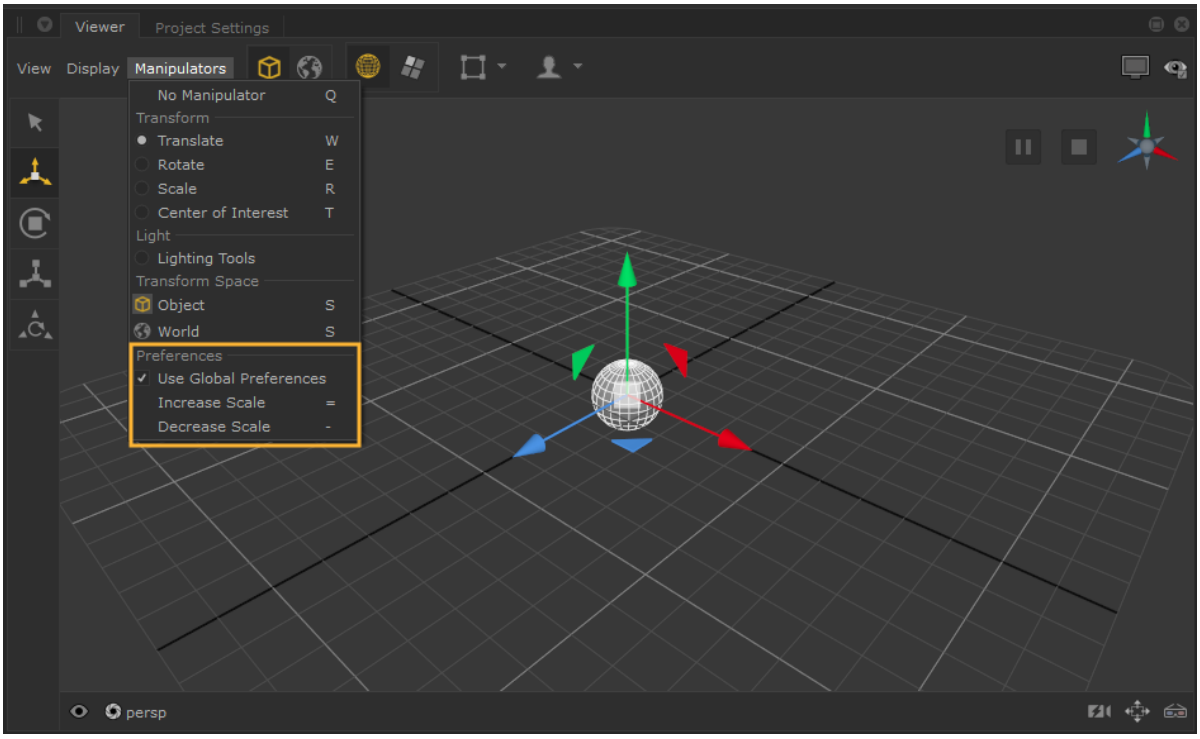
Translation and rotation occurs relative to the world coordinate system.

Manipulator Preferences

In Katana's main menu, open **Edit > Preferences** and select **viewer**. There are some parameters for adjusting manipulator settings:

- **continuousManipulation** - Enables continuous manipulation of manipulators beyond the Viewer boundaries - enabled by default. In the case of remote desktops, disabling this option may be desirable.
- **manipulationFreezePeriod** - The Hydra 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.
- **manipulators > scale** - Adjust the size of the manipulators. This can also be modified interactively within the **Viewer** tab, by activating an option under **Manipulators > Use Global Preferences**.

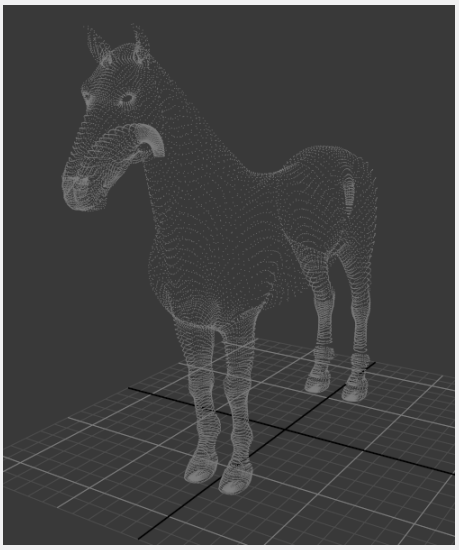
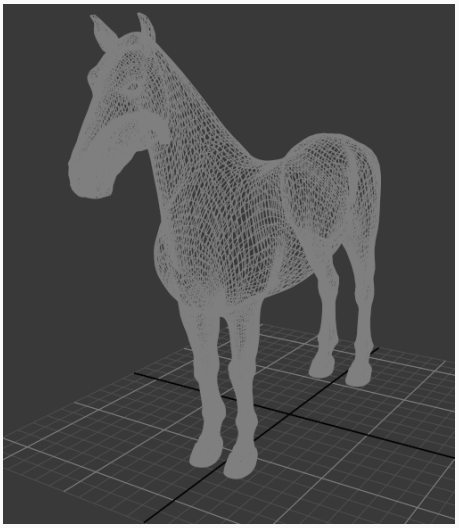
The **Viewer** tab supports scaling transform manipulators using the = and - keyboard shortcuts. This will temporarily scale the manipulators up (=) or down (-).

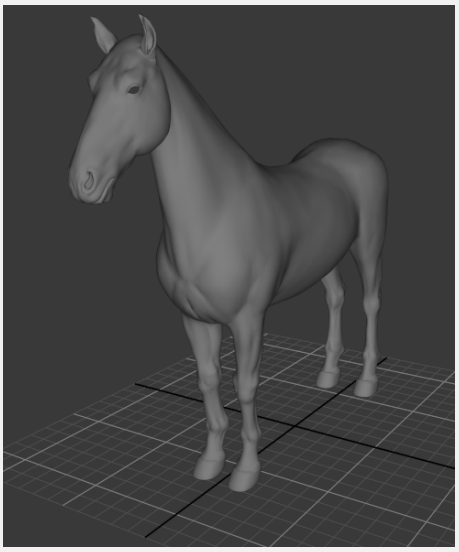
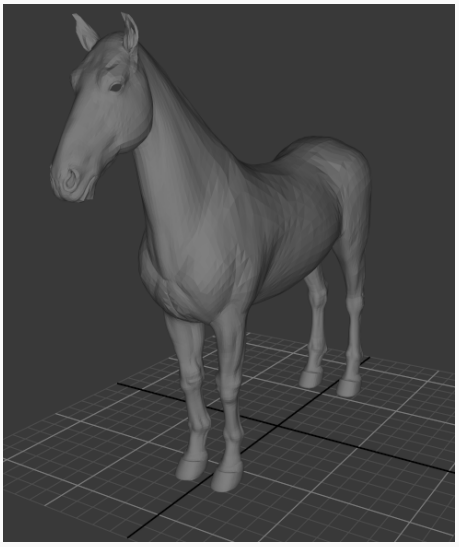


If the option under **Manipulators > Use Global Preferences** is active, any changes to the manipulator size will be synced with the global settings under **Edit > Preferences > viewer > manipulators > scale**. This will scale the manipulators across all Viewer tabs and allow any further changes to persist in future sessions.

Geometry Display Options

To change the overall shading model, select **View** > **[shading model]**:

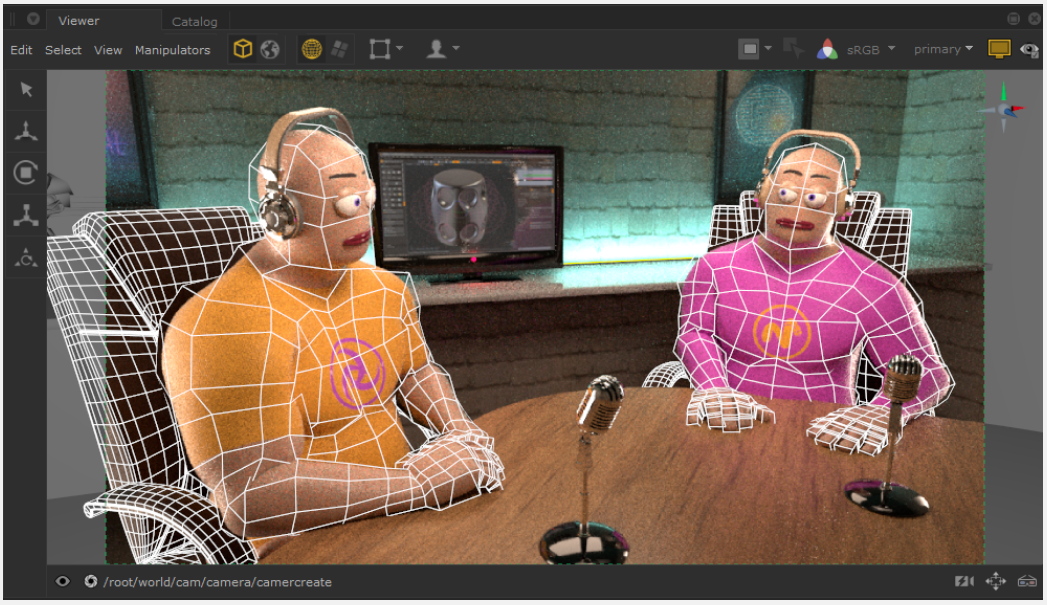
Viewer Behavior	Example
• **Points** - this displays the current 3D scene with each vertex (or control point for a NURBS patch) as a point.	
• **Wireframe** - this displays the current 3D scene with each edge (or surface curve for a NURBS patch) as a line.	

Viewer Behavior	Example
• **Solid** - this displays the current 3D scene with a very simple shader, ignoring scene lights and shadows.	
• **Flat Shaded** - this displays the faceted mesh. Each poly is shaded the same for its normal, whereas **Solid** averages out the values between neighboring polygons to draw a smooth transition.	

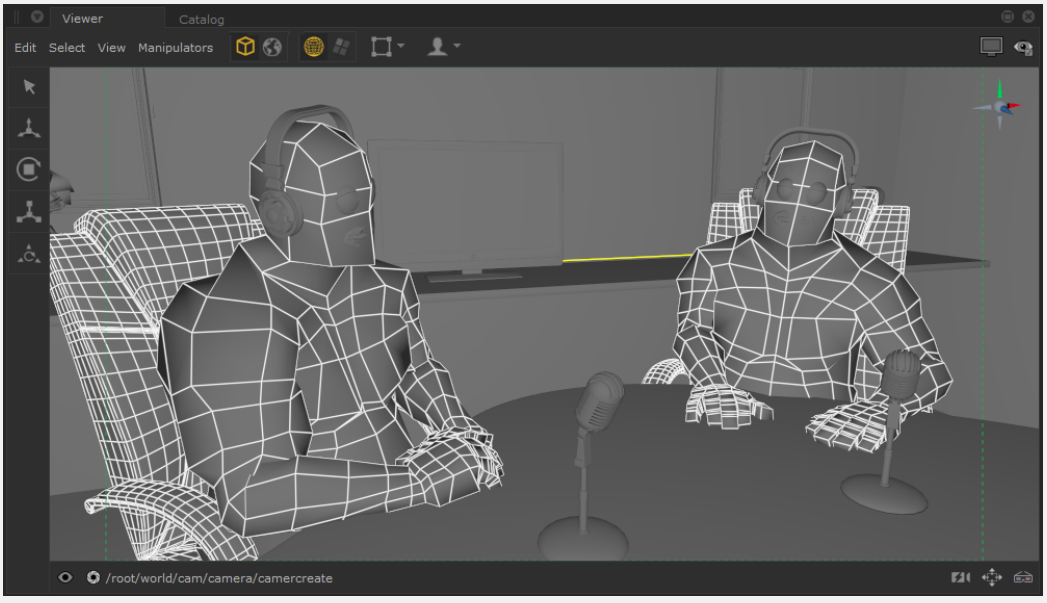
The Monitor Layer in the Hydra Viewer

The Monitor Layer is a new feature of Katana 3.5 which allows you to toggle a render view directly in the Hydra Viewer, overlaying the geometry. This feature is especially useful when working with live renders, as your render overlay will be constantly updated as you make changes to the scene or as you zoom, pan and

rotate within the viewer. This allows for more precision while you are working, as you don't need to switch back and forth between the Monitor tab and the Viewer tab.



Monitor Layer Turned On



Monitor Layer Turned Off

While the Monitor Layer is turned on, you can still make selections within the viewer and the selection feedback is integrated with the rendered image.

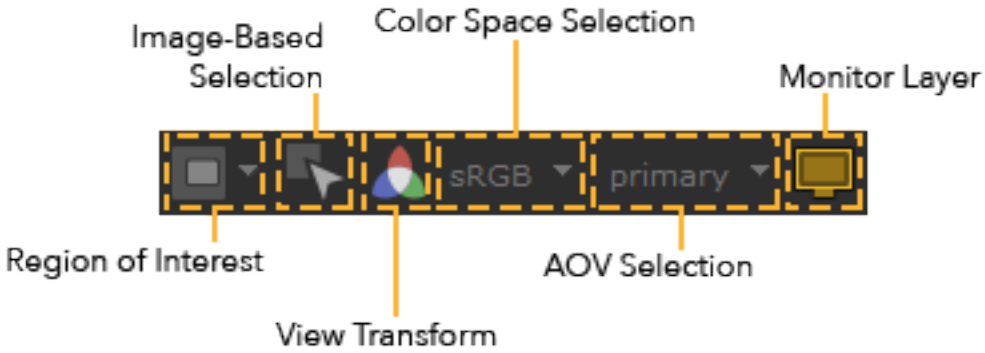
The Monitor Layer creates a smoother workflow for artists, as you can maximize the Viewer tab and have a more focused UI, giving full editing control.

Using the Monitor Layer in the Hydra Viewer

To turn on the Monitor Layer in the Viewer tab, click the Monitor Layer button  , or press ` on the keyboard.

> **Note:** You must have started a render for the overlay to appear. For more information about rendering, see Performing a Render.

Once turned on, the following additional controls are shown in the Viewer tab:



- **Monitor Layer** button - Use this button to toggle the Monitor Layer on and off. When on the button will appear yellow  , and when off the button will appear gray  .
- **AOV selection button** - Use this dropdown to select which AOV to display in the Monitor Layer.

> **Note:** You can cycle through the AOVs in the Monitor Layer using **Shift+Page Down** to move through the list from first to last, and **Shift+Page Up** to move from last to first. You can also use **Shift+Home** to toggle between the default AOV and a previously selected AOV.

- **Color Space** selection button - Use this dropdown to select which color space you would like to view your render in.
- **View Transform** toggle button - Use this button to toggle on and off the view transform to the selected **Color Space**.

- **Image-Based Selection** button - Use this button to toggle Image-Based Selection mode. For more information, see Image-Based Selection in the Monitor Layer.
- **Region of Interest** - Use this to set a region of interest to render. For more information, see Rendering a Region of Interest (ROI).

Image-Based Selection in the Monitor Layer

Scenes rendered in Katana include scene graph location information for each pixel in the rendered image. Image-Based Selection allows you to access this information when viewing a render from the Monitor Layer within the Hydra Viewer.



Note: Information for Image-Based Selection is generated by default when rendering in Katana. You can turn this off by going to:
Edit > **Preferences** > **monitor** and disabling **renderIDPass**.

This can be very useful especially when working with existing renders stored in the **Catalog** tab. While working in the **Node Graph**, you may want to access an object's scene graph location to construct CEL statements, without having to load any heavy geometry.

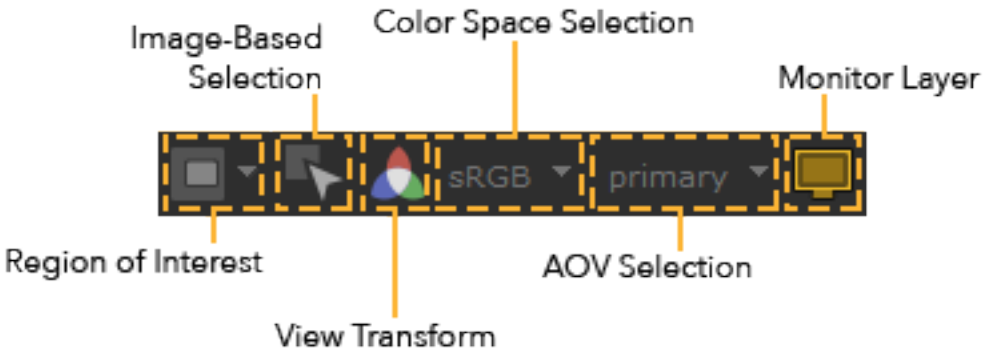
How to Use Image-Based Selection

Image-Based Selection is activated from the Viewer tab and is only available when the Monitor Layer is enabled and a render is loaded.

1. Click the **Monitor Layer** button  , or press ` (backtick) on the keyboard to activate the Monitor Layer.



The following additional controls are shown in the Viewer tab:



Note: For more information about the Monitor Layer, see The Monitor Layer in the Hydra Viewer.

2. Click the **Image-Based Selection** button  or press **I** on the keyboard to activate Image-Based Selection. You can also press and hold the **I** key to temporarily activate Image-Based Selection.

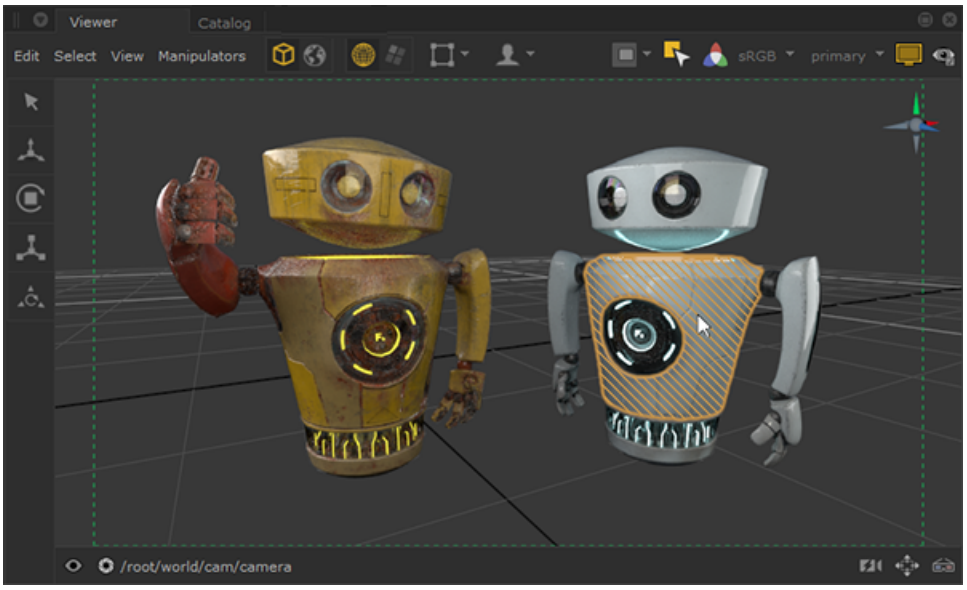


You can now interact with the image and you can no longer select geometry. The geometry does not need to be loaded in the scene for you to make a selection using Image-Based Selection mode.

The hover selection is drawn with a yellow outline.

Note: The color used for highlighting the target selection can be customized by going to: **Edit** > **Preferences** > **viewerHydra** > **monitorLayer** > **highlightColor**

3. **Left-click** to make a selection. For more selection controls, see the Selection Methods section of this topic.

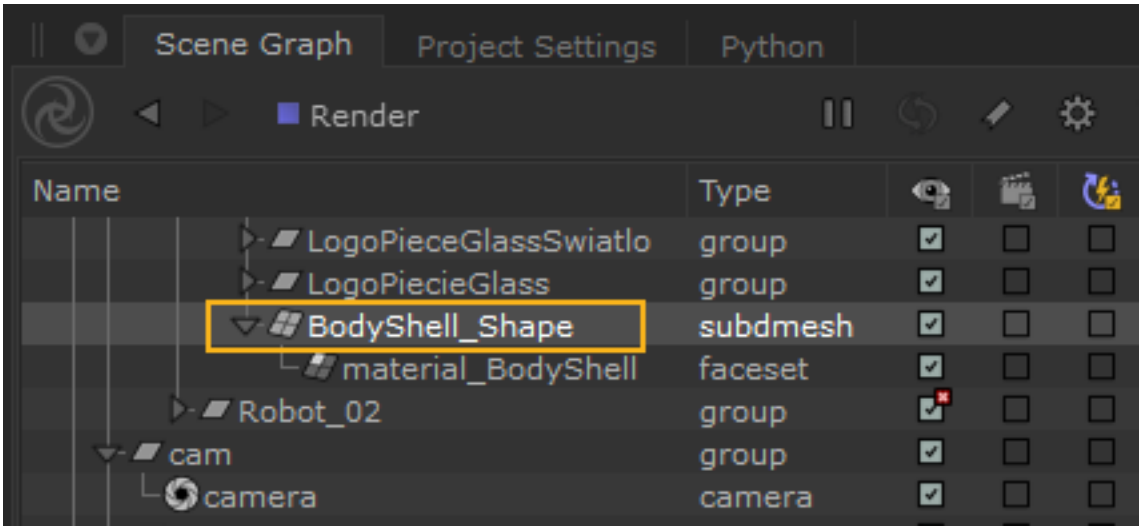


The selection is drawn with a yellow outline and diagonal hatching.

Note: The color used for highlighting the target selection can be customized by going to: **Edit** > **Preferences** > **viewerHydra** > **monitorLayer** > **selectionColor**

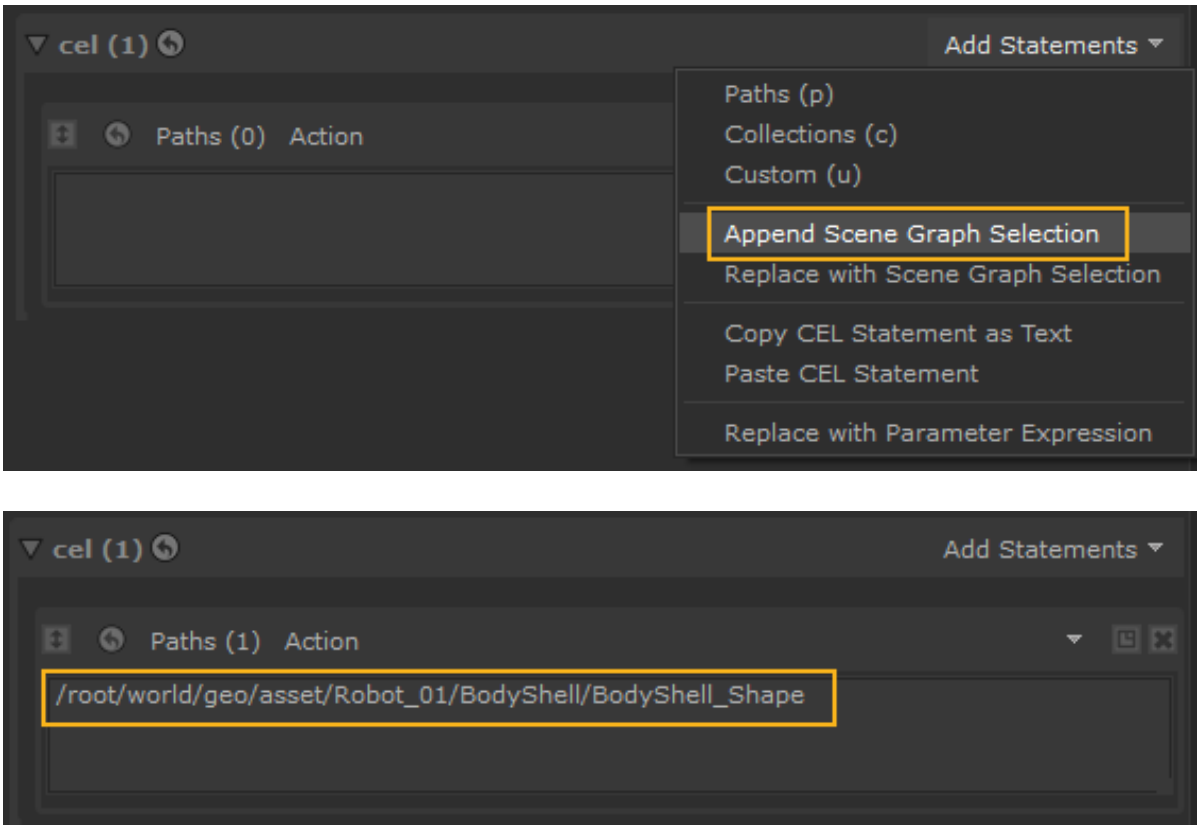
The scene graph location is selected in the Scene Graph if the selected geometry is loaded in the scene.

Note: If the selected geometry is not loaded in the scene, you can press **Ctrl** + **E** over the **Viewer** to expand the locations in the Scene Graph.



This scene graph selection can be used in CEL widgets through **Add Statements** > **Append Scene Graph Selection** / **Replace with Scene Graph Selection**.

Note: The geometry does not have to be loaded in the scene to use the **Append Scene Graph Selection** and **Replace with Scene Graph Selection** options.



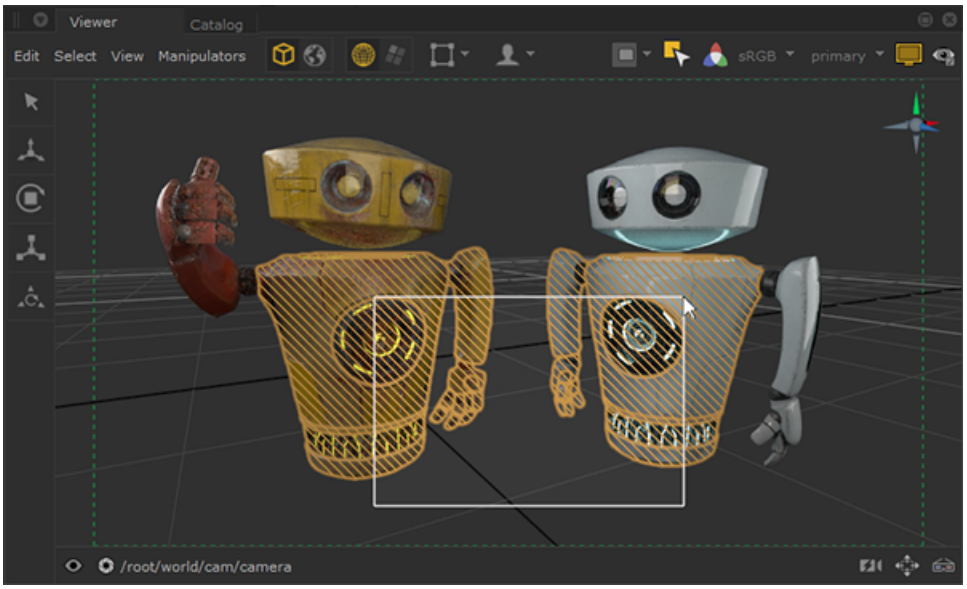
Single scene graph location from selection

Note: For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the Scene Graph Location Widget Type section of the Common Parameter Widgets topic.

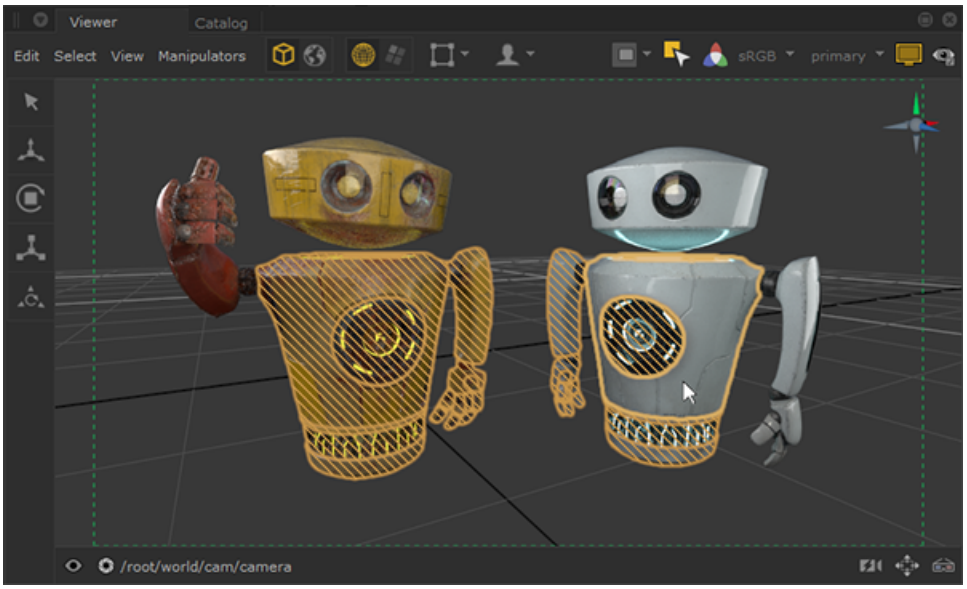
Selection Methods

When Image-Based Selection is active, selections can be made in a variety of ways:

- **Left-Click** - Single selection.
- **Left-Click** + **drag** - Marquee selection.

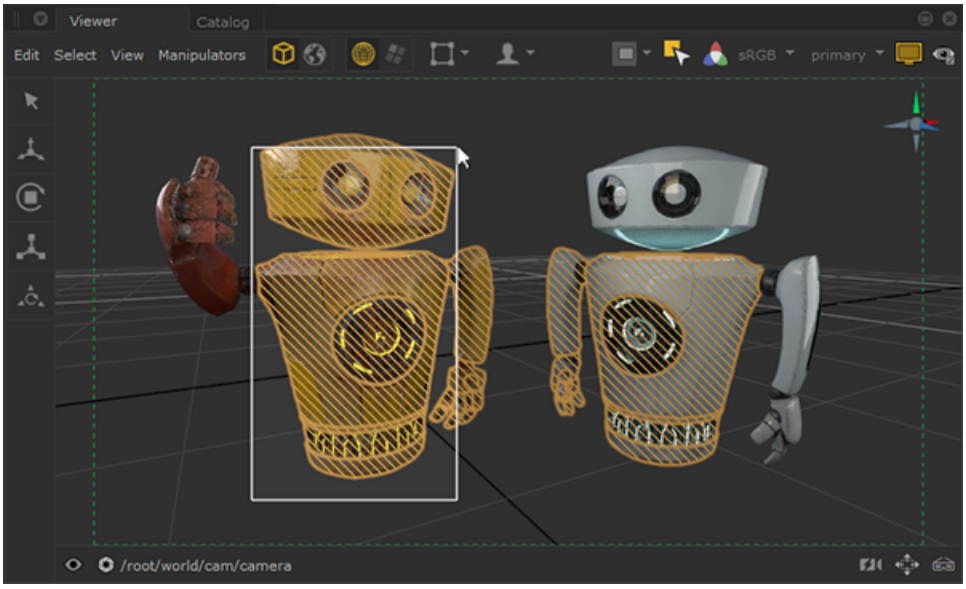


- **Shift** + **Left-Click** / **Shift** + **Left-Click** + **drag** - Toggle selections.
  This inverts your selection.
- **Ctrl**+ **Left-Click** / **Ctrl**+ **Left-Click** + **drag** - Remove selections.



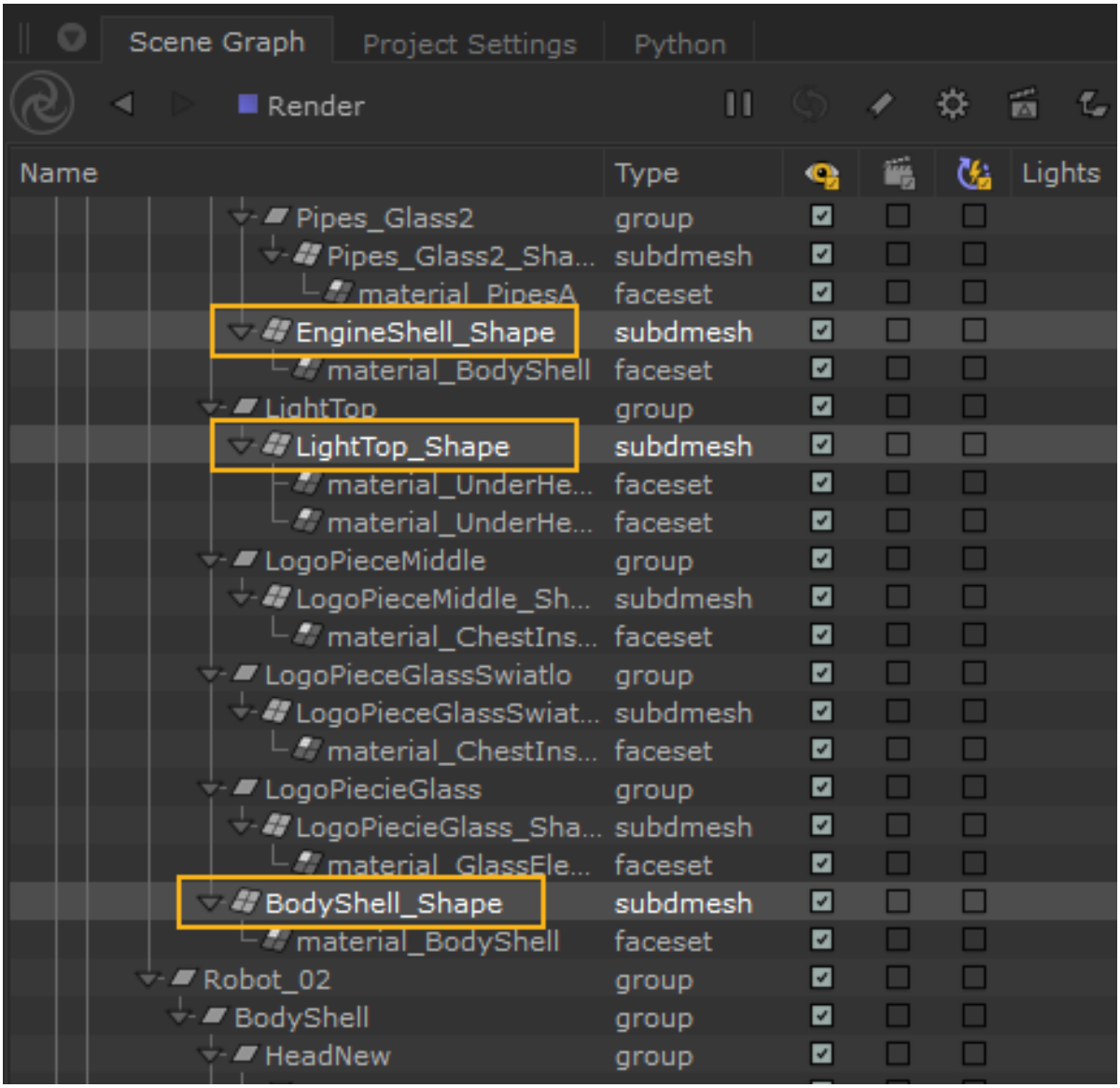
- **Ctrl**+**Shift** + **Left-Click** / **Ctrl**+**Shift** + **Left-Click** + **drag** - Append selections.

  This adds to your selection.

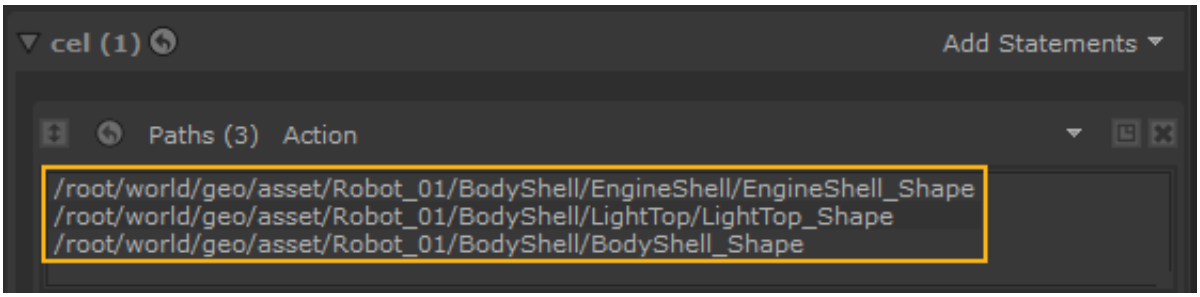


- Click on an empty area to deselect everything.

Multiple scene graph location selections can be used in CEL widgets in the same way, through **Add Statements** > **Append Scene Graph Selection** / **Replace with Scene Graph Selection**.



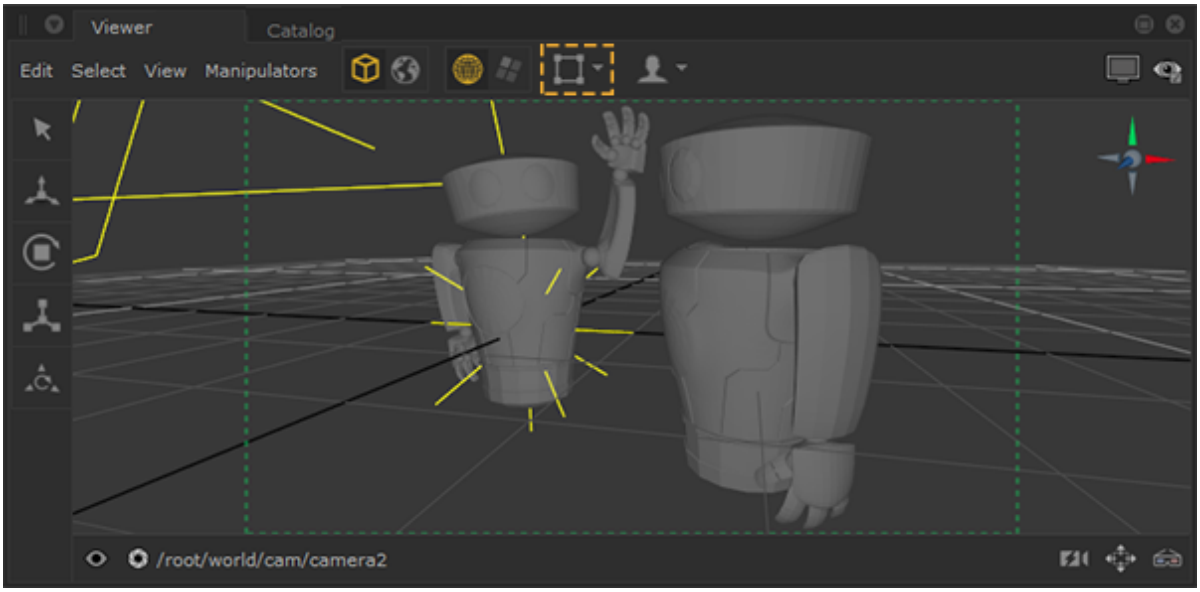
Multiple selections in Scene Graph



Multiple scene graph locations in CEL widget

Snapping

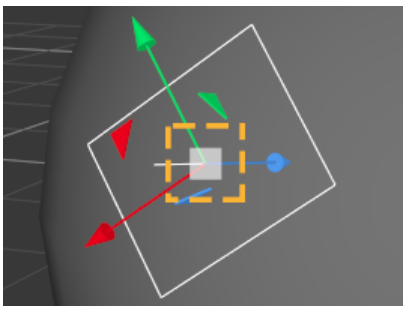
Snapping in the Hydra Viewer allows you to snap an object's Translate or Center of Interest manipulator to a target location's individual Vertices, Edges, Faces, Center or Object, as well as to Lights, Cameras, Locators, and Rigs. Snapping can be turned on and off using the Snapping tool button, or by pressing the V key. Snapping can also be turned on temporarily by holding the V key.

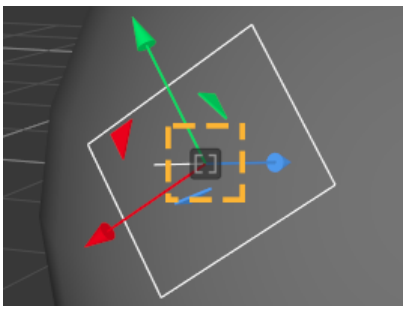


When Snapping is turned on the button appears yellow  , and when turned off the button appears gray  .

Note: The symbol on the button changes depending on the chosen Snapping mode.

When Snapping is activated, this is also indicated on the manipulator handle.





Default handle Handle with Snapping on

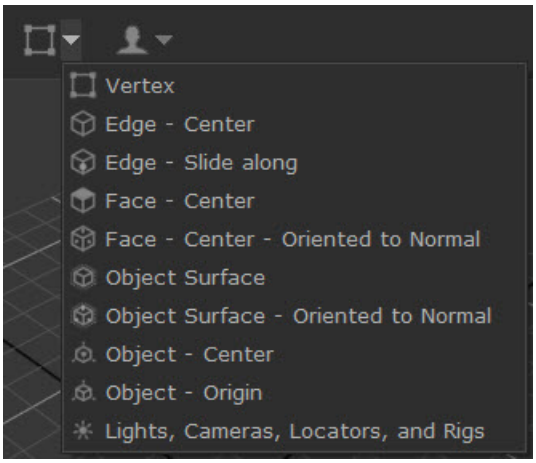
While the object's manipulator snaps to the given position, the onscreen mouse position is indicated by a cross and square brackets, known as the hit area.



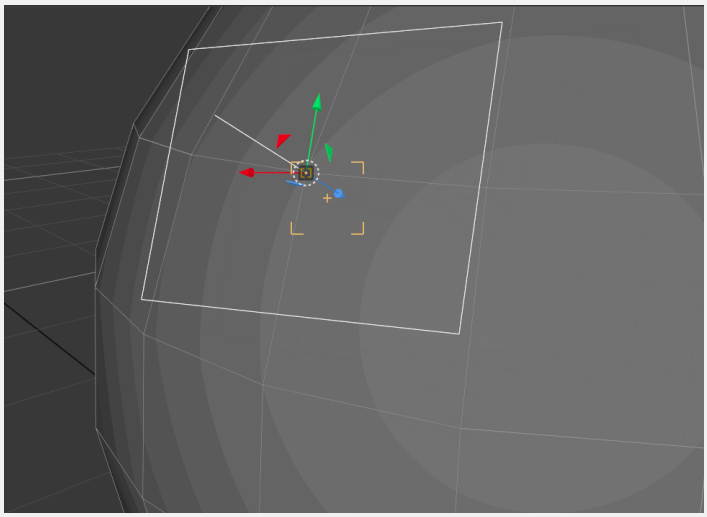
Mouse position

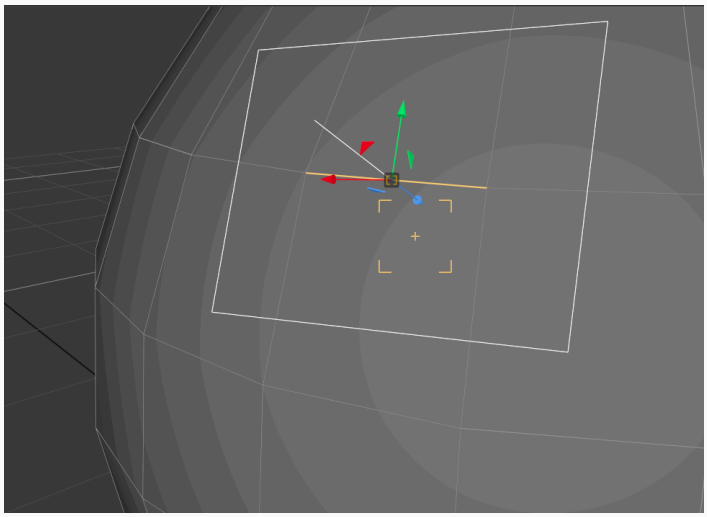
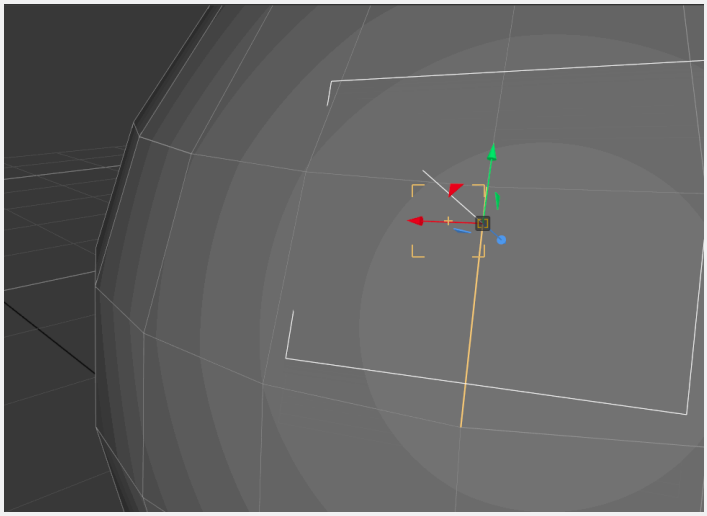
Note: Specify the size of the hit area by going to:
Edit > **Preferences** > **viewer** > **snapping**

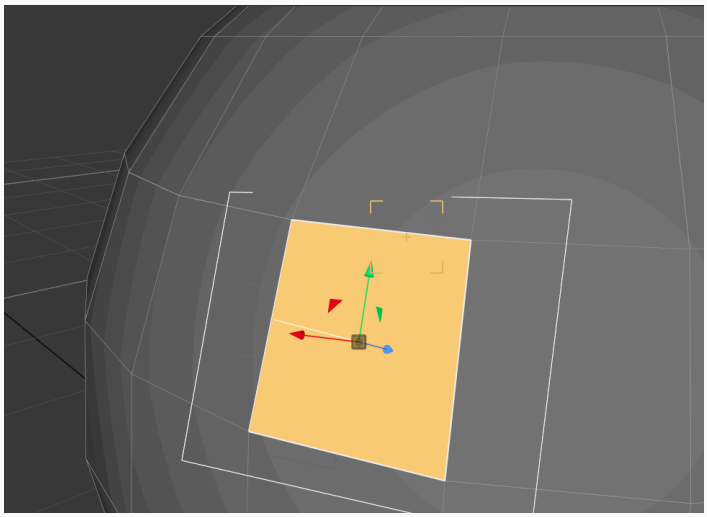
The type of snapping mode can be selected from the dropdown menu. You can use Shift + V to cycle through the available snapping mode options.

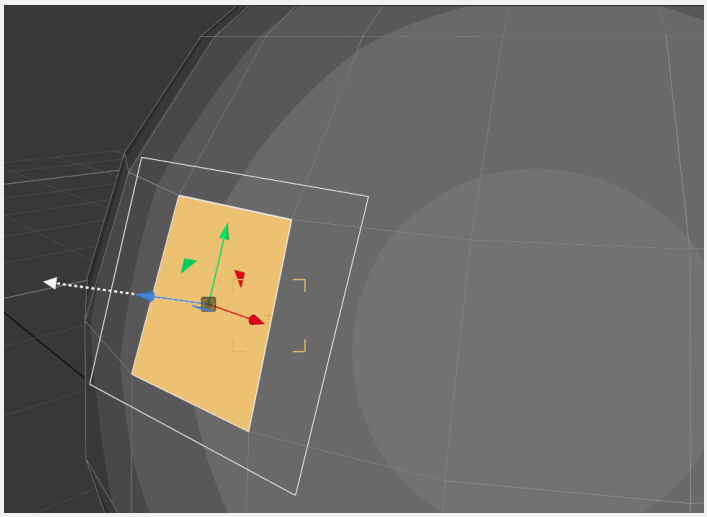
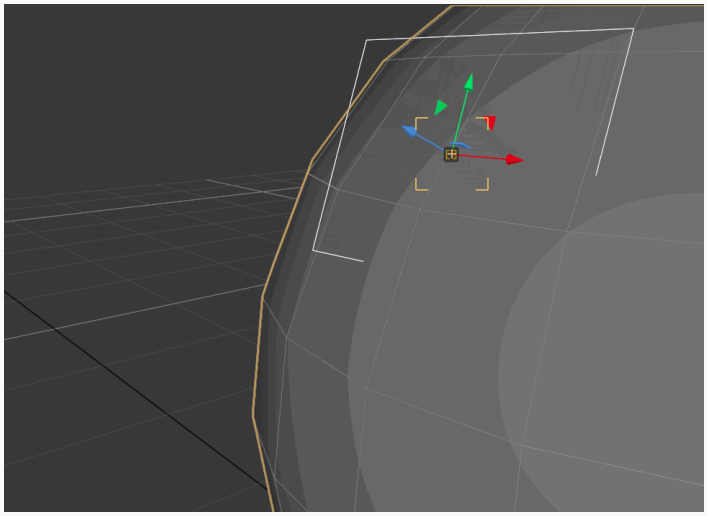


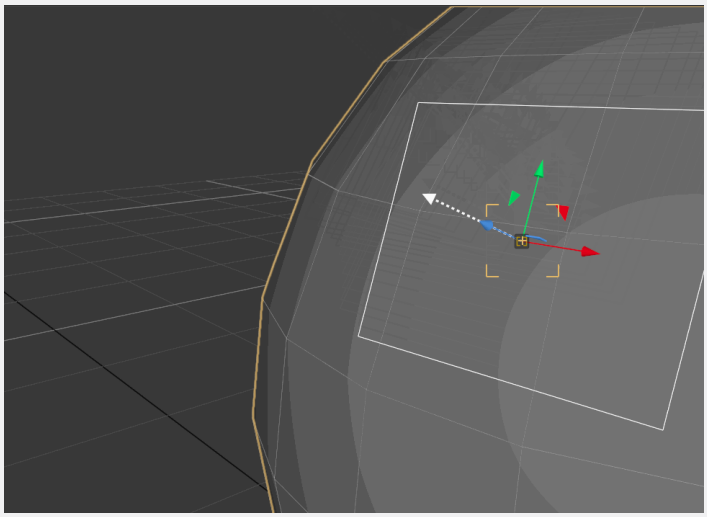
Snapping Modes

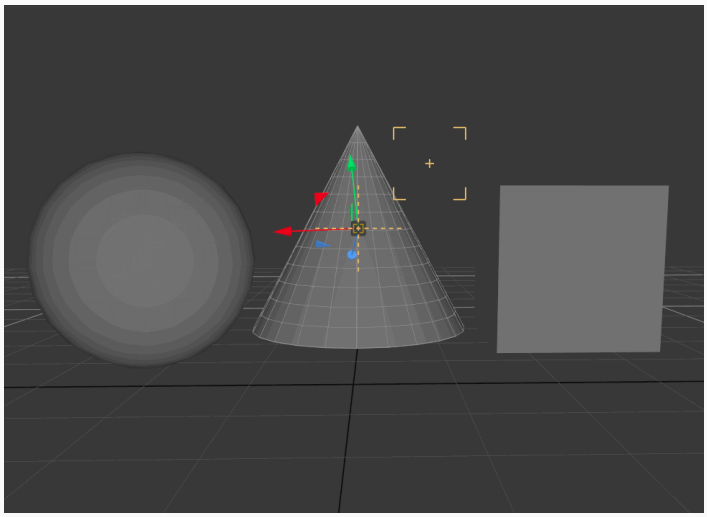
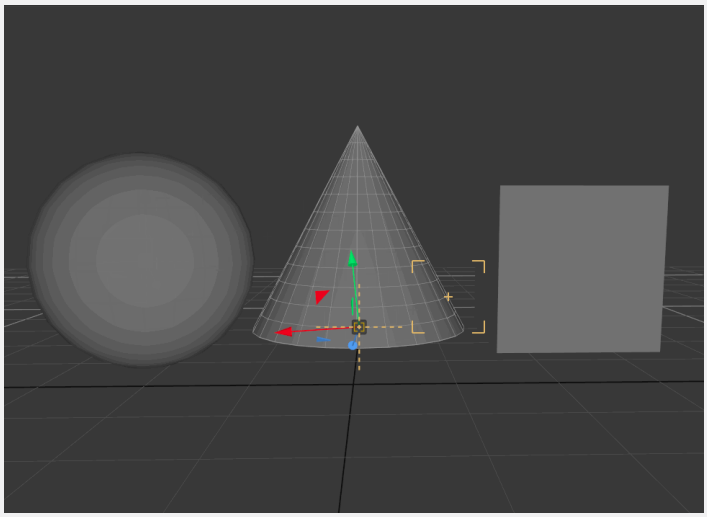
Symbol	Mode	Description
	Vertex	Snap to individual vertices. The target vertex is outlined and highlighted, in addition to the target mesh's snapping wireframe. **Note:** Only vertices of front-facing faces are considered. 
	Edge - Center	Snap to the center of an edge. The target edge is highlighted in addition to the target mesh's snapping wireframe.

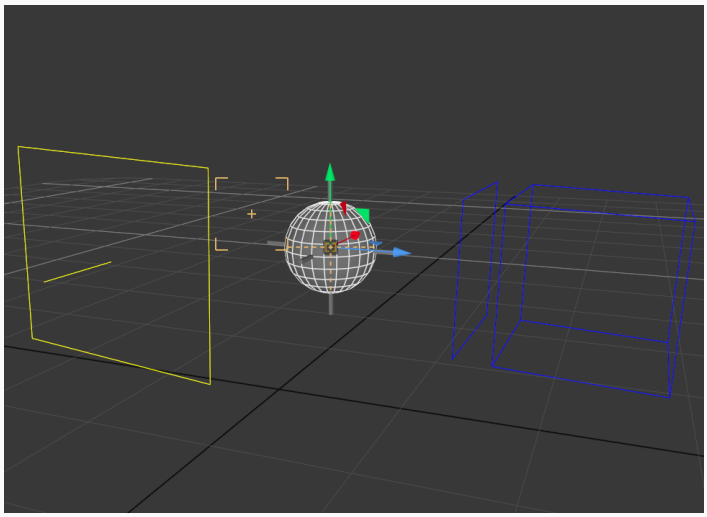
Symbol	Mode	Description
		
	Edge - Slide Along	Snap to, and slide along an edge. The target edge is highlighted in addition to the target mesh's snapping wireframe. 
	Face - Center	Snap to the center of target face. The target face is highlighted in addition to the target mesh's snapping wireframe.

Symbol	Mode	Description
		
	Face - Center - Oriented to Normal	Snap to the center of a given face and orient the transformed mesh's rotation to point along the normal of the target face. The normal is indicated by a white arrow with a dashed line. The target face is highlighted in addition to the target mesh's snapping wireframe.  **Note:** In Oriented to Normal modes, the **OrientAxis** and **UpAxis** can be specified at: **Edit** > **Preferences** > **viewerHydra** > **snapping**

Symbol	Mode	Description
		
	Object Surface	Snap to the closest point on an object's surface. The target mesh is drawn with an outline, in addition to the target mesh's snapping wireframe being drawn. 
	Object Surface - Oriented to Normal	Snap to the closest point on an object's surface and orient the transformed mesh's rotation to point along the normal of the closest face. The normal is indicated by a white arrow with a dashed line. The target mesh is drawn with an outline, in addition to the target

Symbol	Mode	Description
		mesh's snapping wireframe being drawn. **Note:** In Oriented to Normal modes, the **OrientAxis** and **UpAxis** can be specified at: **Edit** > **Preferences** > **viewerHydra** > **snapping** 
	Object - Center	Snap to the center of an object, defined by a bounding box automatically generated by the extents of the geometry. The center is highlighted with a crosshair icon. The target mesh's wireframe is highlighted.

Symbol	Mode	Description
		
	Object Origin	Snap to the origin of an object. The origin is highlighted with a cross-hair icon. The target mesh's wireframe is highlighted. 
	Lights, Cameras, Locators, and Rigs	Snap to the center of any light, camera, locator, or rig in your scene. Geometry in the scene is be ignored by this mode. The center is highlighted with a cross-hair icon.

Symbol	Mode	Description
		

Note: The color used for drawing the target mesh's snapping wireframe can be customized by going to:
Edit > **Preferences** > **viewer** > **snapping** > **wireframeColor**

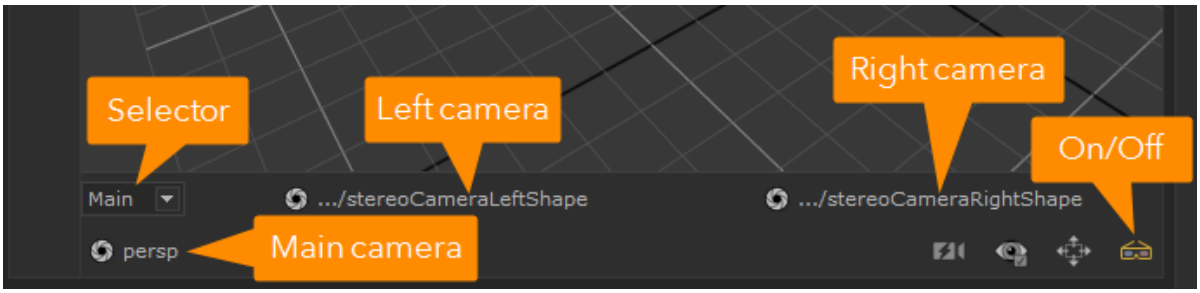
Using Stereo Cameras in the Hydra Viewer

When designing scenes for stereo displays, you may need to preview each side of the view (left eye/right eye) to make appropriate adjustments. To support this, the Hydra Viewer has stereo mode controls. These allow you to assign a left and right camera in addition to the main camera. You can then switch the viewpoint between your main, left, and right cameras.

Note: To use stereo mode you must have a rig with the left and right cameras in place. For example, you can add and name cameras using CameraCreate nodes, or you can import a pre-configured rig.

Using the Stereo Controls

1. Click the stereo controls toggle to reveal the controls.



2. The selector on the left (labeled 'Selector' in the diagram above) chooses the view camera. The selector has three options: **Main**, **Left**, and **Right**.

   Left and **Right** correspond to the two cameras that are selected in the left and right camera dropdowns to the right of the camera selector. Click on these dropdowns to change the left and/or right camera selection.

   The **Main** camera corresponds to the one shown in the camera selector (labeled 'Main camera' in the diagram) at the bottom-left of the viewer.

Subdivision and Anti-Aliasing in the Hydra Viewer

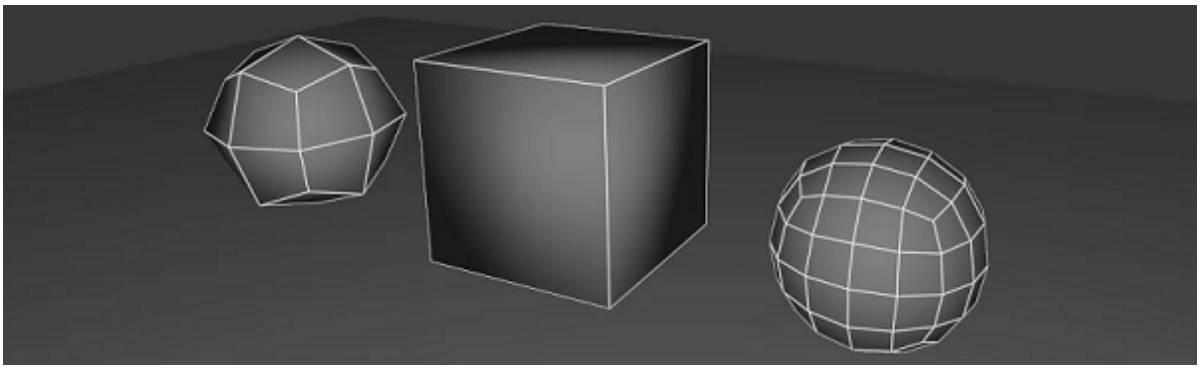
Subdivision

The Hydra Viewer provides three levels of subdivision (0, 1, 2) for the geometry on display.

Using Subdivision in the Hydra Viewer

1. Select the geometry using the scene graph or viewer.

2. In the **Viewer** tab, Open **Edit** > **Set Subdivision Level**
3. Select the subdivision level.
4. Alternatively use the shortcut keys **0**, **1**, and **2** to set the subdivision level.
5. You can also check the active subdivision level by selecting an object and opening the menu.



Cubes at various levels of subdivision.

Note: This is only applicable to subdmesh types, so polymesh types will need to be converted first.

Anti-Aliasing in the Hydra Viewer

The Hydra Viewer offers four levels of Multi-Sample Antialsing. For more information see Specific Viewer Behavior.

The OpenGL driver for the graphics card returns a value for GL_MAX_SAMPLES. This value indicates the maximum supported number of samples for multi-sampling you can have.

To adjust the MSAA, select **Edit > Preferences** and click **hydraViewer**, adjust the **antiAliasing** setting:

- **Full** - the full value of the maximum number of supported samples.
- **Medium** - half the value of the maximum number of supported number of samples.
- **Low** - quarter of the value of maximum number of samples
- **Off** - MSAA is disabled.

Note: When you open a new Viewer tab, the number of samples in use for MSAA is reported on the Katana console.

Live Rendering with the Hydra Viewer

The Hydra Viewer features a Monitor Layer, enabling you to view a render over the top of your scene displayed in the Hydra Viewer.

For a full overview of this feature, please refer to The Monitor Layer topic.



Note: For background information, see the following pages:
Using the Monitor Tab
Controlling Live Rendering
Controlling Live Rendering in the Scene Graph

Live Render from Viewer Camera

Once a Live Render has been started, you can use the **Live Render from Viewer Camera** button  to change the active Hydra Viewer camera to the camera that is being used in the live render. This means that the camera defined in the RenderSettings node when can be overridden to the active Hydra Viewer camera.

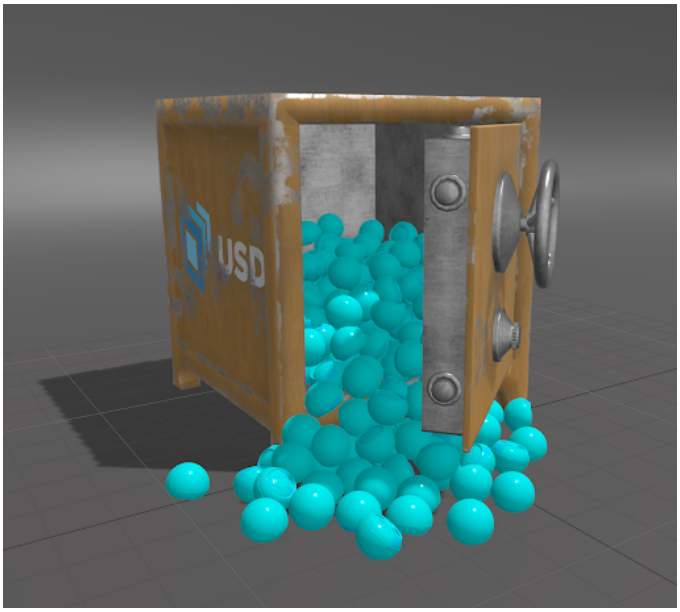


Note: The **Live Render from Viewer Camera** button  changes the render camera but does not start a live render, there must already be an active live render for this button to have an effect.

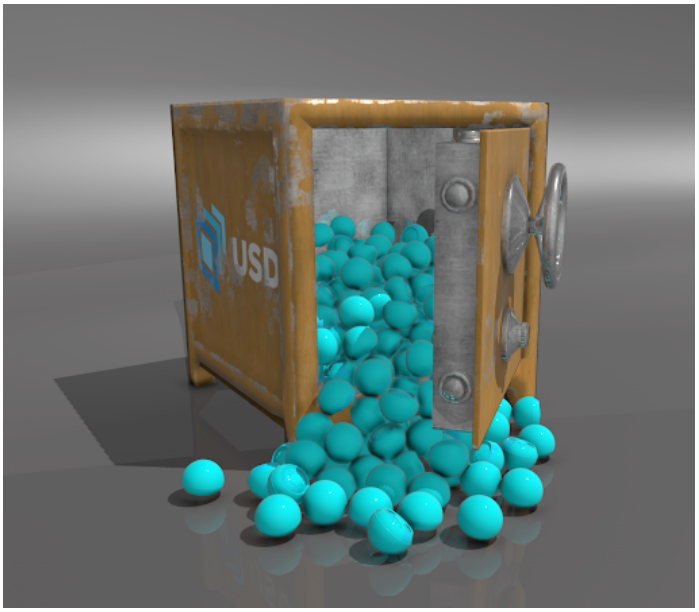
When starting a Live Render, only the camera is added to the Live Render Working Set which means that only changes made to the camera trigger updates to the Live Render. If you want the Live Render to update after adjustments to other locations, these locations must be added to the Live Render Updates Working Set by checking the appropriate box in the Live Render Updates  working set of the scene graph.
For more information, see Controlling Live Rendering in the Scene Graph.

Render Delegates in the Hydra Viewer

Render Delegates are a way to preview lighting, materials, and assets within the Hydra Viewer in the context of your renderer. Delegates allow you to get a richer preview of your work, without the need to set a render, and are fully customizable meaning you have control over the look and optimization of your delegate.

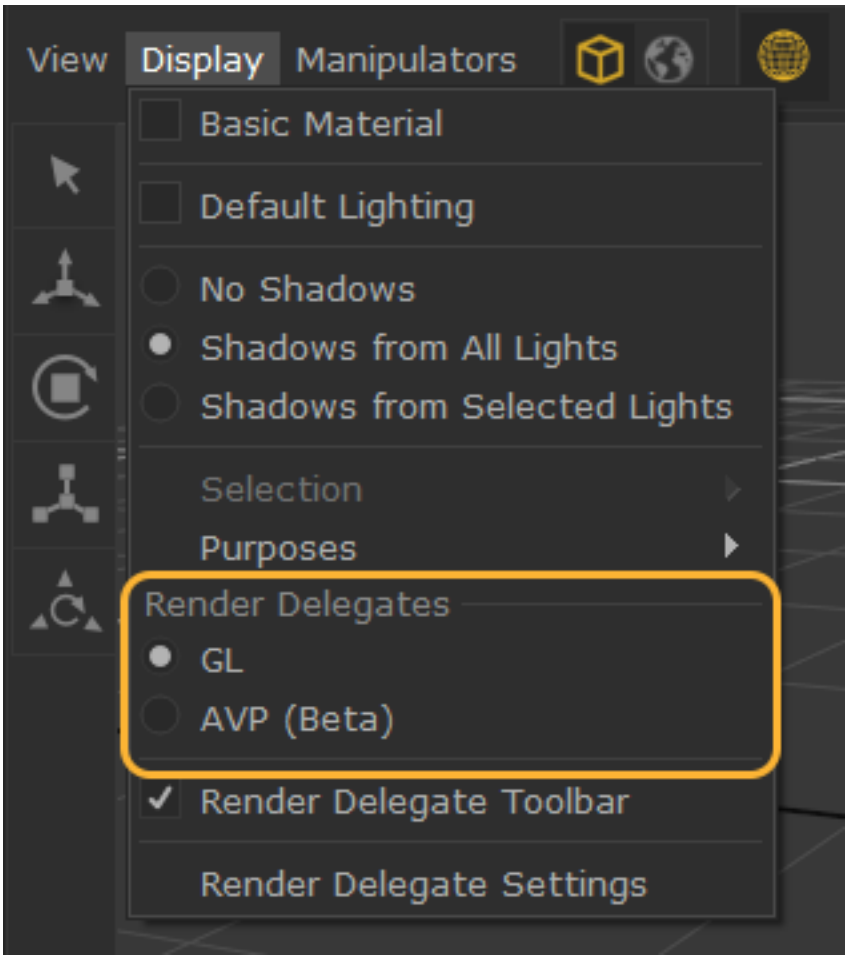


GL Render Delegate (default)



AVP Render Delegate

Katana currently ships with the Advanced Viewport (AVP) as an out-of-the-box GL delegate and can be accessed under the **Render Delegate** section of the **Viewer** tab.



Third-party render vendors that support Hydra Delegates can also be used within Katana, but their plugins will need to be loaded into Katana before they can be used.

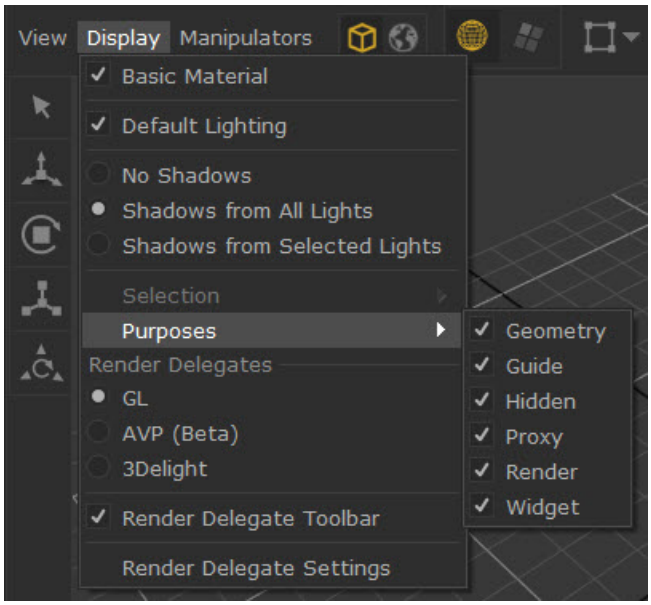
Note: For more information about loading plugins into Katana, and creating a launcher script, refer to these Support articles:
Linux: Creating a Katana Launcher Script for Linux
Windows: Creating a Katana Launcher Script for Windows

When using delegates belonging to third-party render vendors, if their shader is present in your project, the delegate automatically defaults to the corresponding shader. For example, a Renderman delegate automatically defaults to using a PxrSurface if it is present in your scene, regardless of any other vendor shaders that may be present. However, render delegates do still support basic materials and default lighting, so renderer-specific shaders are not a necessity.

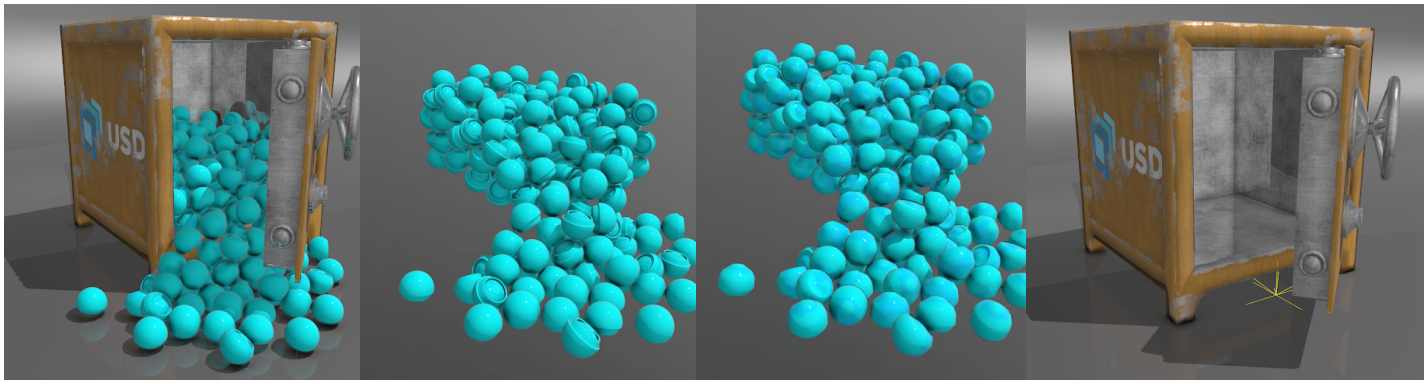
Viewing USD Purposes

Within delegates, USD files with purposes set up can have their purposes enabled or disabled within the viewer. This allows you to pick and choose what you can and can't see in the viewer and is useful for optimization.

Purposes can be enabled and disabled by navigating to **Display** > **Purposes** and toggling the tick box next to the name of each purpose.



Enabling or disabling purposes are useful for scene optimization. Certain purposes like Render tend to contain higher resolution assets and are heavier on processing as a result. Because of this, disabling Render purposes and enabling something like Proxy or Guide allows you to work on a scene without having to sacrifice performance.



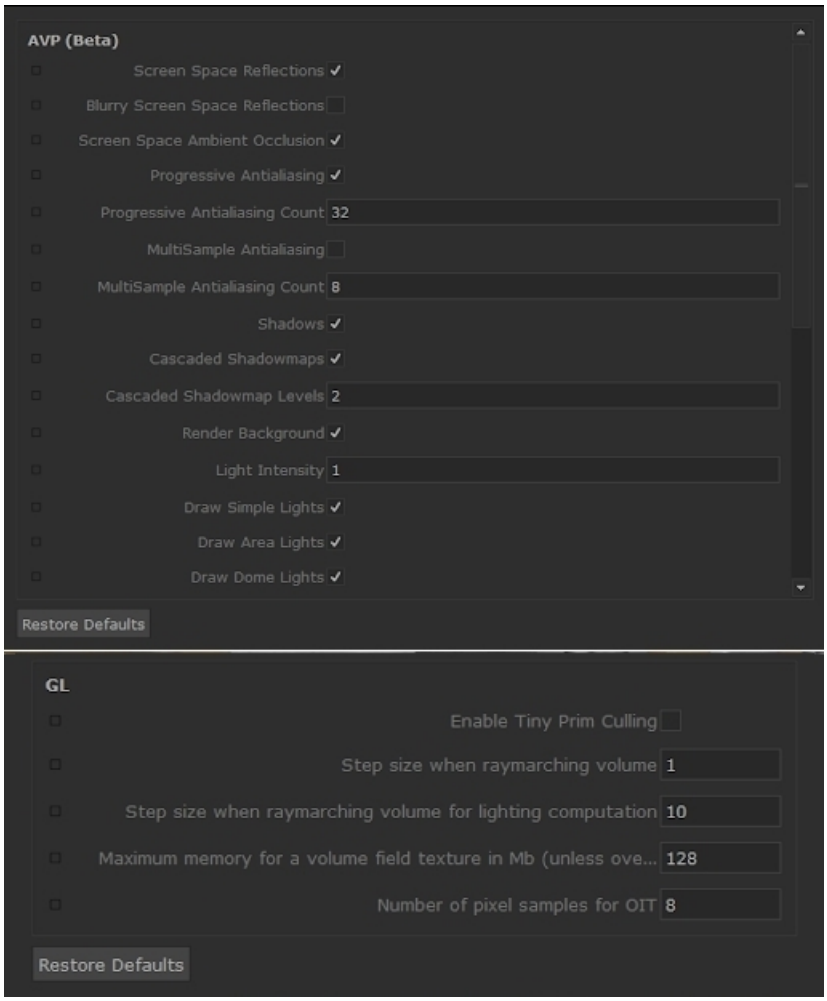
All purpose | Only Render purpose | Only Proxy purpose | Only Geometry purpose



Note: For more information on USD Purposes, see USD Terms and Concepts: Purpose.

Customizing Your Delegate

Delegates can be customized by accessing the **Render Delegate Settings** window, available in the **Display** tab. The settings available in the **Render Delegate Settings** window will be dependant on the delegate set to that renderer. For example, when set to GL, the settings will ve specific to the GL delegate, while settings specific to AVP will be available when AVP is set.

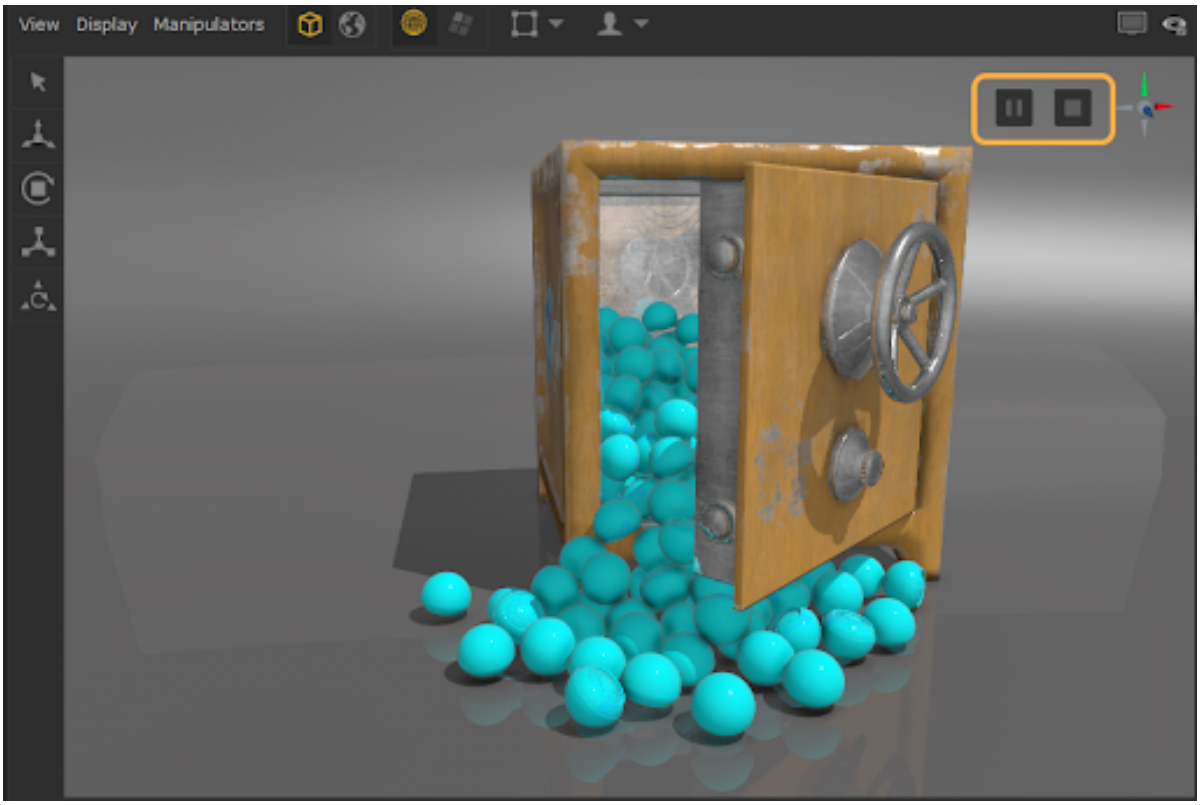


Delegate Settings for AVP and GL

Within the settings window, you can adjust the quality of the render in the viewer, as well as enable or disable reflections, shadows, and ambient occlusion.

Delegates can also be paused or stopped at any time. The stop/pause setting can be enabled or disabled by toggling **Display** > **Render Delegates Toolbar**.

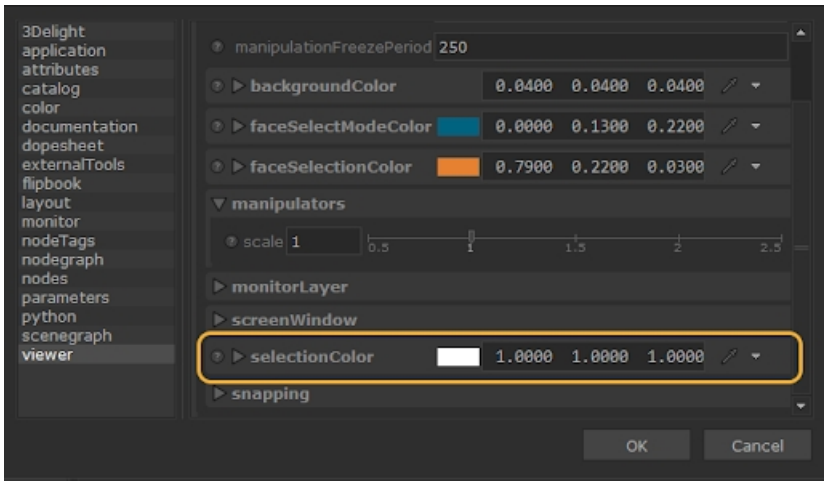
Render delegates will only support stop or pause, but not both. Depending on the delegate, Katana will dynamically call the option that is supported and make it available to you, while the unsupported option is disabled. However, as stop/pause is not supported by GL rendering, these options will not be available when using the AVP or GL delegates.



Being able to pause and stop your delegate is useful, as it allows you to continue to work on heavy scenes without having to sacrifice performance. Once edits are made, you can unpause the delegate and any changes will be reflected in the viewer.

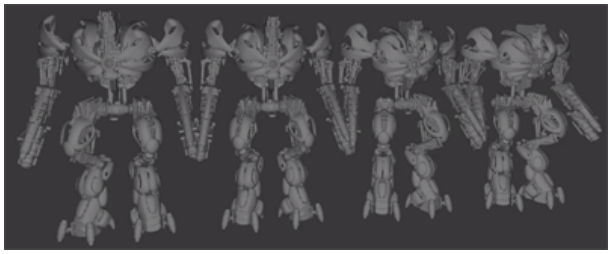
You are also able to modify how selected elements are highlighted within the Viewer when using GL renderer delegates, making selections easier to see. The selection color is also customizable, and can be changed by navigating to **Edit** > **Preferences**> **Viewer** and adjusting the parameter available under **selectionColor**.

At this time, standard selection modes and their customizable features are not supported when working with non-GL render delegates.

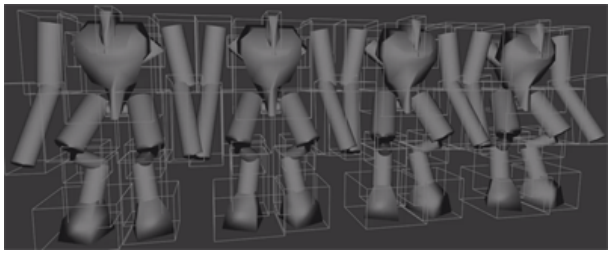


Proxies and Bounding Boxes

Proxy geometry acts as a simplified stand-in for more complex geometry, allowing you to work with the scene graph in the viewer without the overhead of complete geometry. Bounding boxes simplify the representation even further by drawing a box around the region occupied by the geometry.



Detailed geometry.



The same geometry represented using proxy geometry and bounding boxes.

The Viewer tab allows you to see any proxies or bounding boxes that have been defined on the geometry.

Video: Watch this video for a quick overview on viewing Proxy geometry and bounding boxes in the Hydra Viewer.

In order to view proxies and bounding boxes, the scene graph must be set to the right level of detail:

1. Collapse the scene graph by right-clicking on the required branch and select **Collapse Branch** or if you're at root, select **Collapse All**.

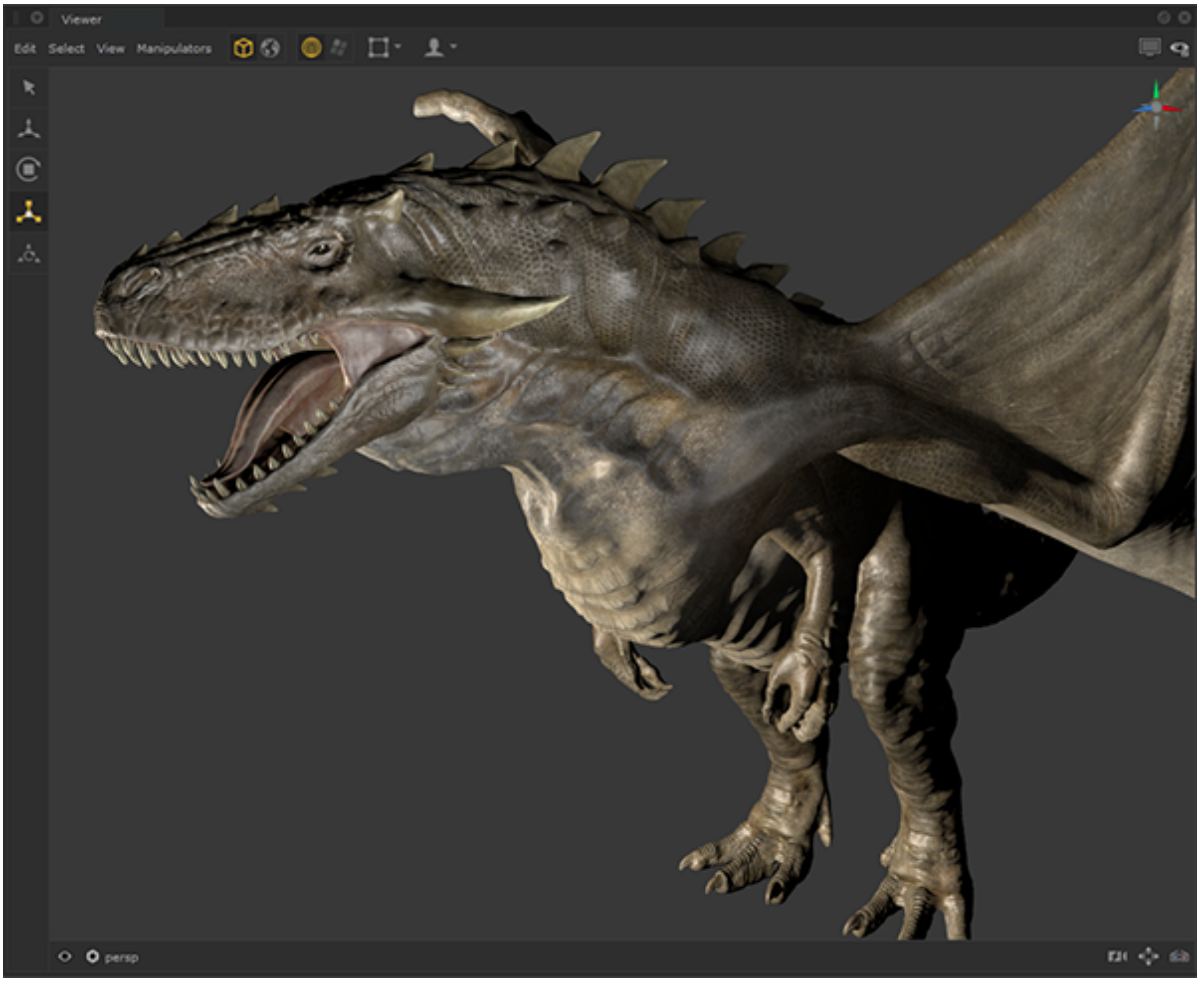
2. Right-click on the branch (or root) select **Expand to and Select Proxy Children**.

To active bounding boxes or proxies in the **Viewer** tab, select **View > Bounding Boxes** or/and **View > Proxy Geometry**.

Displaying Textures in the Hydra Viewer

The Hydra Viewer in Katana allows you to view geometry loaded in your scene. By default, the geometry is drawn using a Hydra shader which doesn't interact with lights in your scene, or provide a preview of your materials.

UsdPreviewSurface is one of many USD shading nodes that allow you to view materials, lights and shadows in the Hydra Viewer. This is very useful as it provides artists with a preview of what their scene looks like without needing to perform a render. It allows you to experiment with material properties, such as roughness or specularity, and immediately see the changes you make in the Hydra Viewer.



Loading USD Plug-ins

Find out how to load the USD plug-ins into Katana.

Setting up USD Materials

Discover how to assign **UsdPreviewSurface** materials to your objects and view them in the Hydra Viewer.

Using USD Lights

Learn how to set up USD lights in your scene and view them in the Hydra Viewer.

Loading USD Plug-ins into Katana

In Katana 4.5v1, and later, USD plug-ins are enabled by default. You don't need to define any environment variables, you can use USD plug-ins straight away.

To use the USD nodes inside versions of Katana earlier than 4.5v1, you must first enable the USD plug-ins so that they are loaded when you open Katana.

To do this, you must edit your **KATANA_RESOURCES**, **LD_LIBRARY_PATH** (**PATH** on Windows), and **PYTHONPATH** in your Katana launcher script and add the USD plugin folder.



Note: For more information about creating a launcher script for Katana, refer to these Support articles:
Linux: Creating a Katana Launcher Script for Linux
Windows: Creating a Katana Launcher Script for Windows

1. Add the following lines to your launcher script:

   Note: **<KATANA_ROOT>** represents the path to your Katana install folder, for example: C:\Program Files\Katana4.0v1

   Windows

   ```
   set PATH=%PATH%;<KATANA_ROOT>\plugins\Resources\Usd\lib
   set KATANA_RESOURCES=%KATANA_RESOURCES%;<KATANA_ROOT>\plugins\Resources\Usd\plugin
   set PYTHONPATH=%PYTHONPATH%;<KATANA_ROOT>\plugins\Resources\Usd\lib\python
   ```

   Linux

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<KATANA_ROOT>/plugins/Resources/Usd/lib
   export KATANA_RESOURCES=$KATANA_RESOURCES:<KATANA_ROOT>/plugins/Resources/Usd/plugin
   export PYTHONPATH=$PYTHONPATH:<KATANA_ROOT>/plugins/Resources/Usd/lib/python
   ```

2. Launch Katana using the launcher script and the additional USD node types are available from the node creation menu.

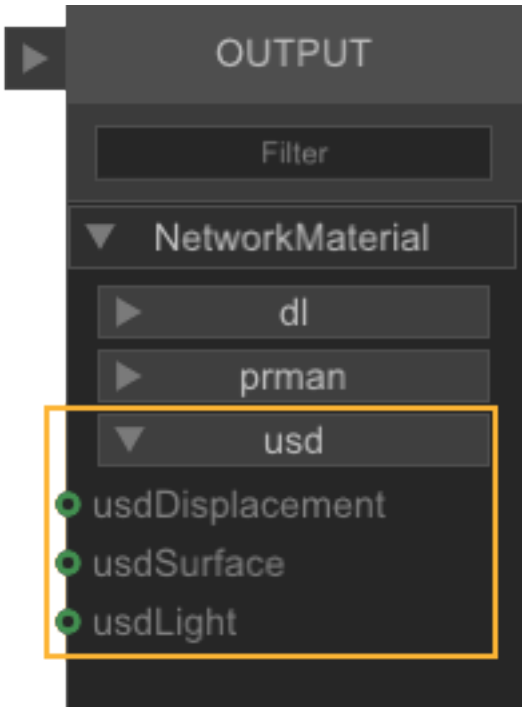


Additional USD nodes



Additional USD shading nodes

A **usd** menu is also loaded on the Terminal sidebar inside NetworkMaterialCreate nodes.



Note: For more information on USD plug-ins for Katana, refer to the Katana USD Plug-ins section in the Developer Guide.

Setting up UsdPreviewSurface Materials

You can use **UsdPreviewSurface** shading nodes to build USD materials. You can then assign these materials to your object and view the result in the Hydra Viewer.

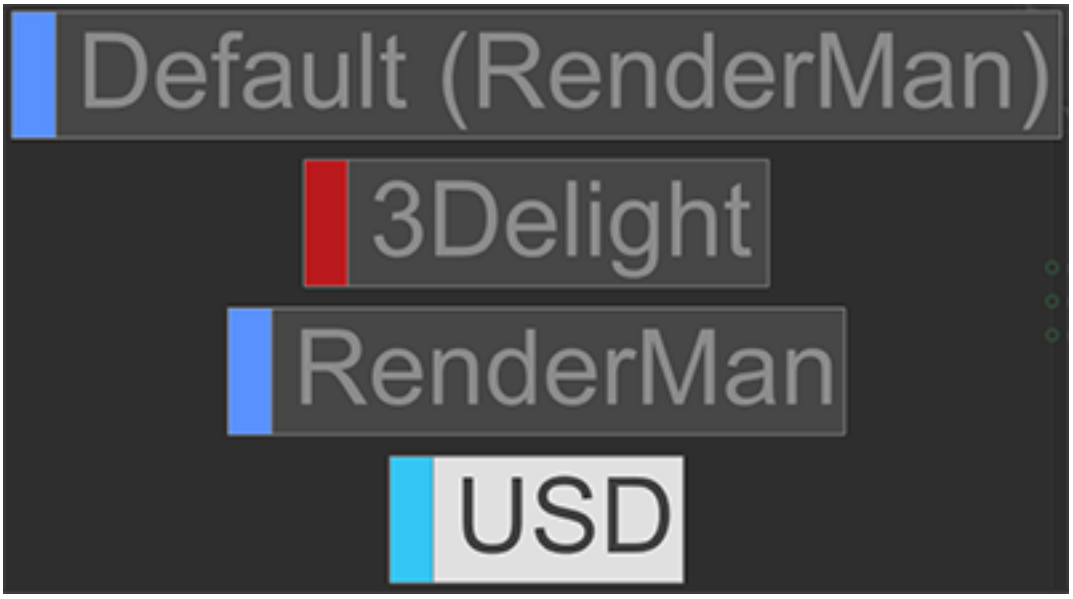
UsdPreviewSurface shading node types are accessible from the **USD** shading node creation menu inside NetworkMaterialCreate nodes.

To use **UsdPreviewSurface** to display materials in the Hydra Viewer:

1. Load the USD plug-ins into Katana by following the steps in the Loading USD Plug-ins into Katana topic.
2. Create a **NetworkMaterialCreate** node and jump inside it, or jump inside an existing **NetworkMaterialCreate** node.

> **Note:** If you are using an existing NetworkMaterialCreate node, you may need to refresh the Sidebar Terminal to see the **usd** outputs. To do this, open the **Parameters** for the **NetworkMaterialCreate** node and choose **Shelf Actions** > **Refresh Sidebar Terminal**.

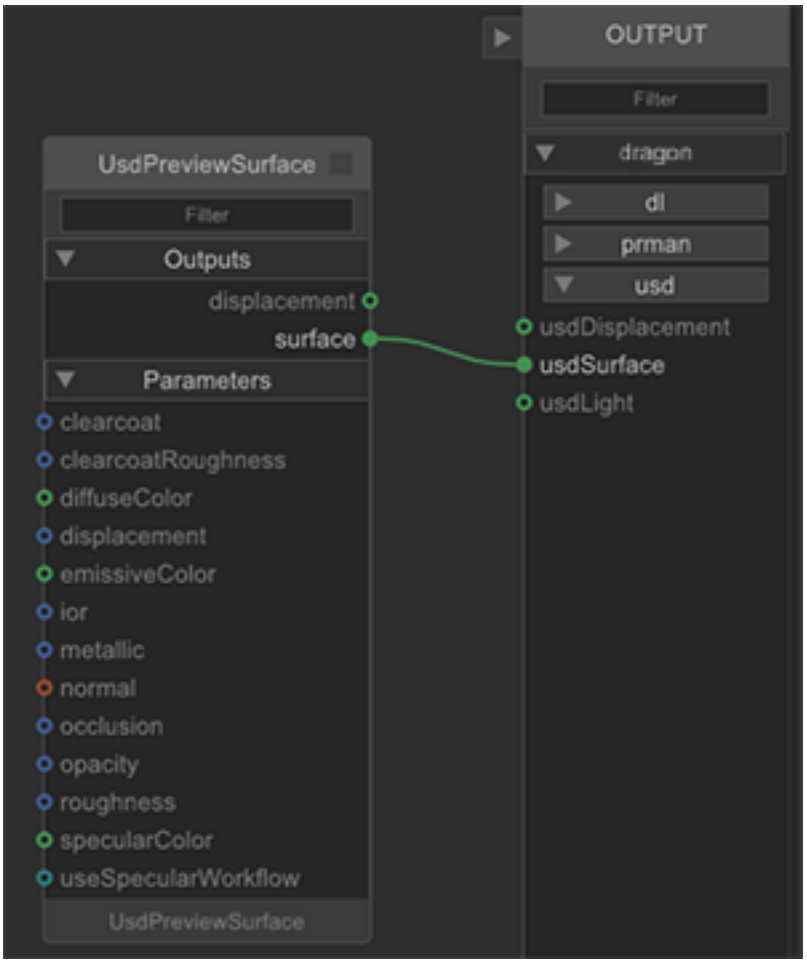
3. Press **Shift** + **Tab** to open the renderer selection menu and choose **USD**.



4. Press **Tab** to open the USD shading node creation menu.
5. Select **UsdPreviewSurface** and place the node.



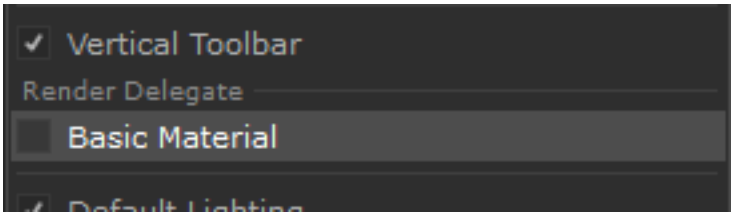
6. Connect the **surface** output from the **UsdPreviewSurface** node to the **usdSurface** input under the **usd** drop-down in the Terminal sidebar.



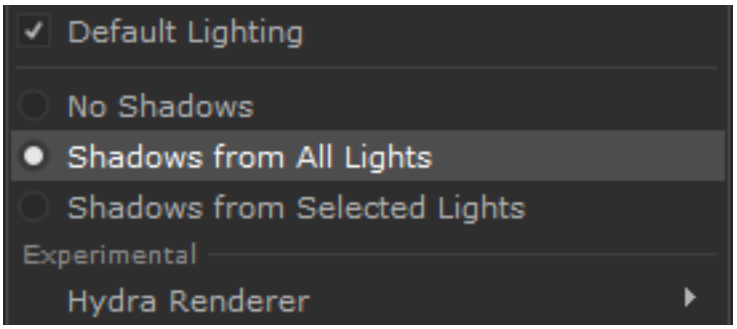
7. Open the **Parameters** for the **UsdPreviewSurface** and make the required adjustments.
8. Use a **MaterialAssign** node to assign the material to your geometry.

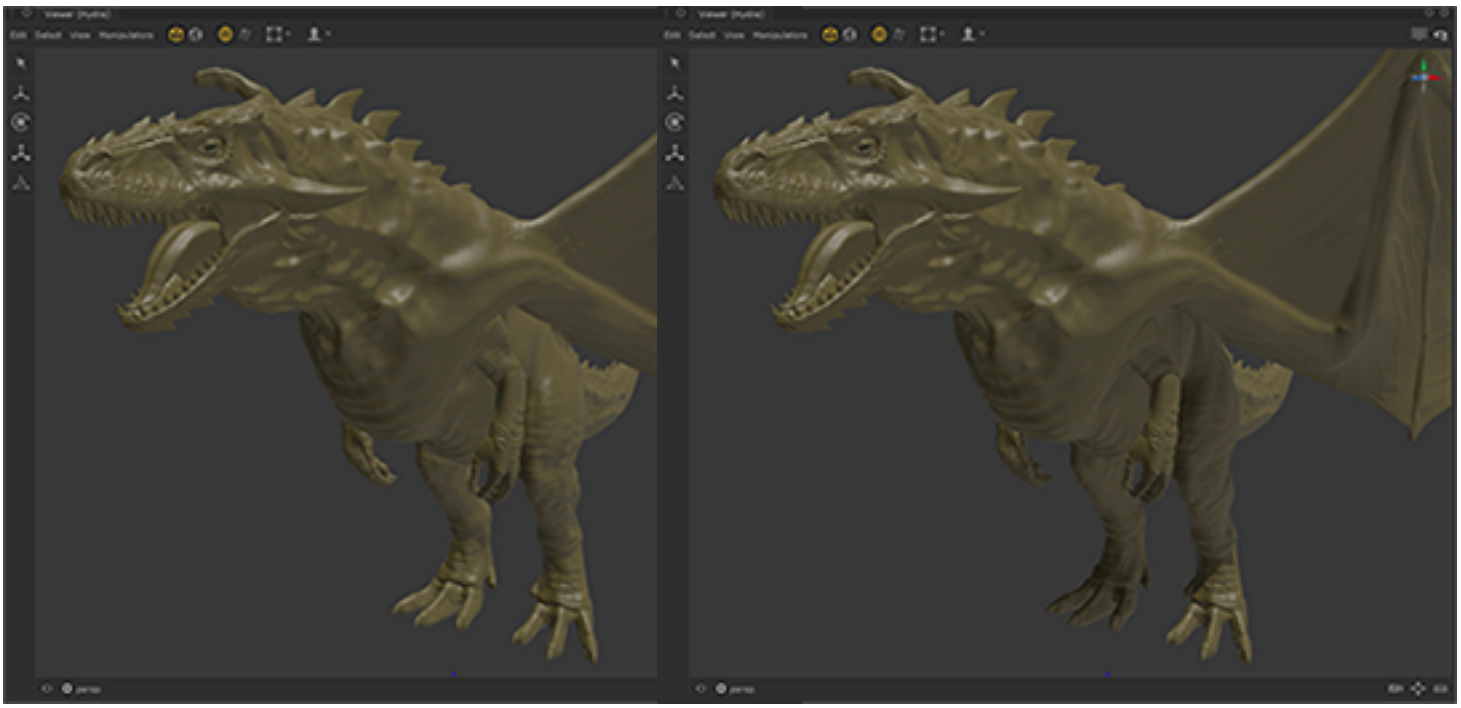
> **Note:** For more information about assigning materials, see the Assigning Materials and Textures section of the Material Basics topic.

9. In the **Viewer** tab, click **View** and disable **Basic Material** to preview your USD material.



In the **Viewer** tab, click **View** and choose **Shadows from All Lights** to preview your material with shadows. If you only want selected lights to cast shadows, you can use the **Shadows from Selected Lights** option.





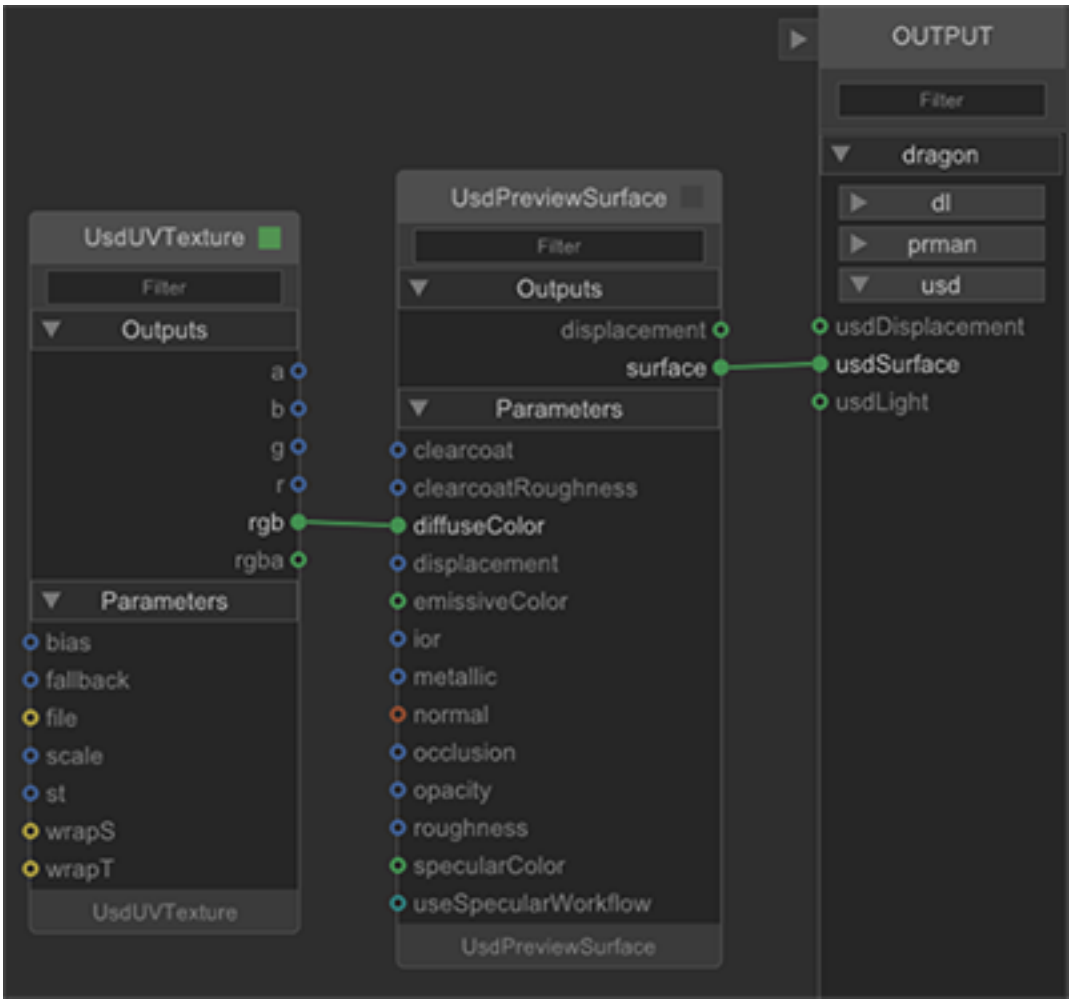
Basic Material off
No Shadows

Basic Material off
Shadows from All Lights

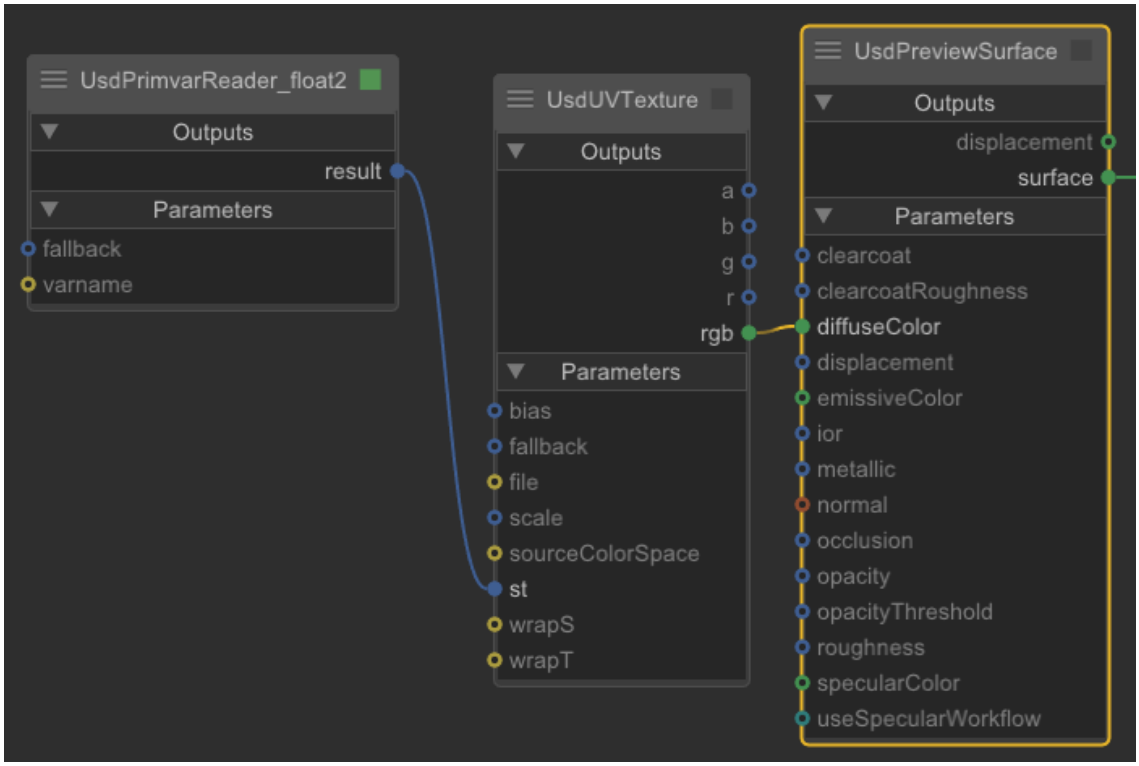
You can assign texture maps to **UsdPreviewSurface** shading nodes using **UsdUVTexture** node types.

To assign a texture map to the **diffuseColor** parameter in your **UsdPreviewSurface** shading node:

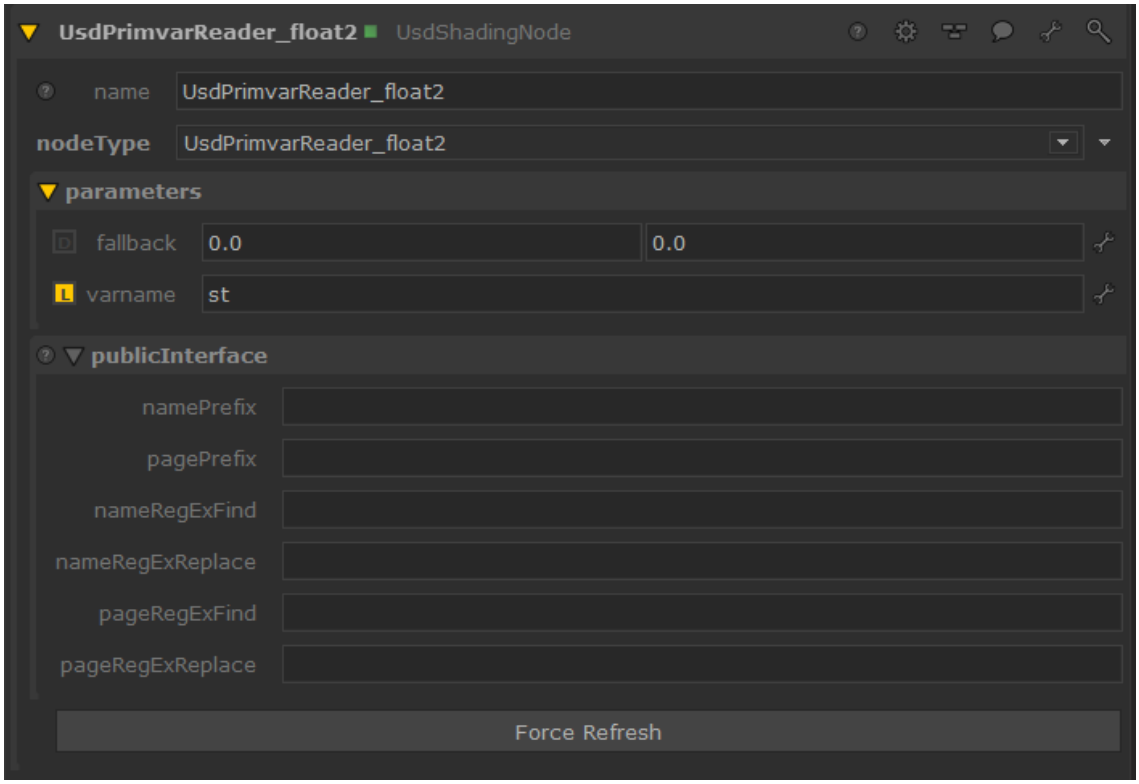
10. Create a **UsdUVTexture** node, open the **Parameters**, and enter the file path in the **file** parameter.
11. Connect the **rgb** output from the **UsdUVTexture** node to the **diffuseColor** input of the **UsdPreviewSurface** node.



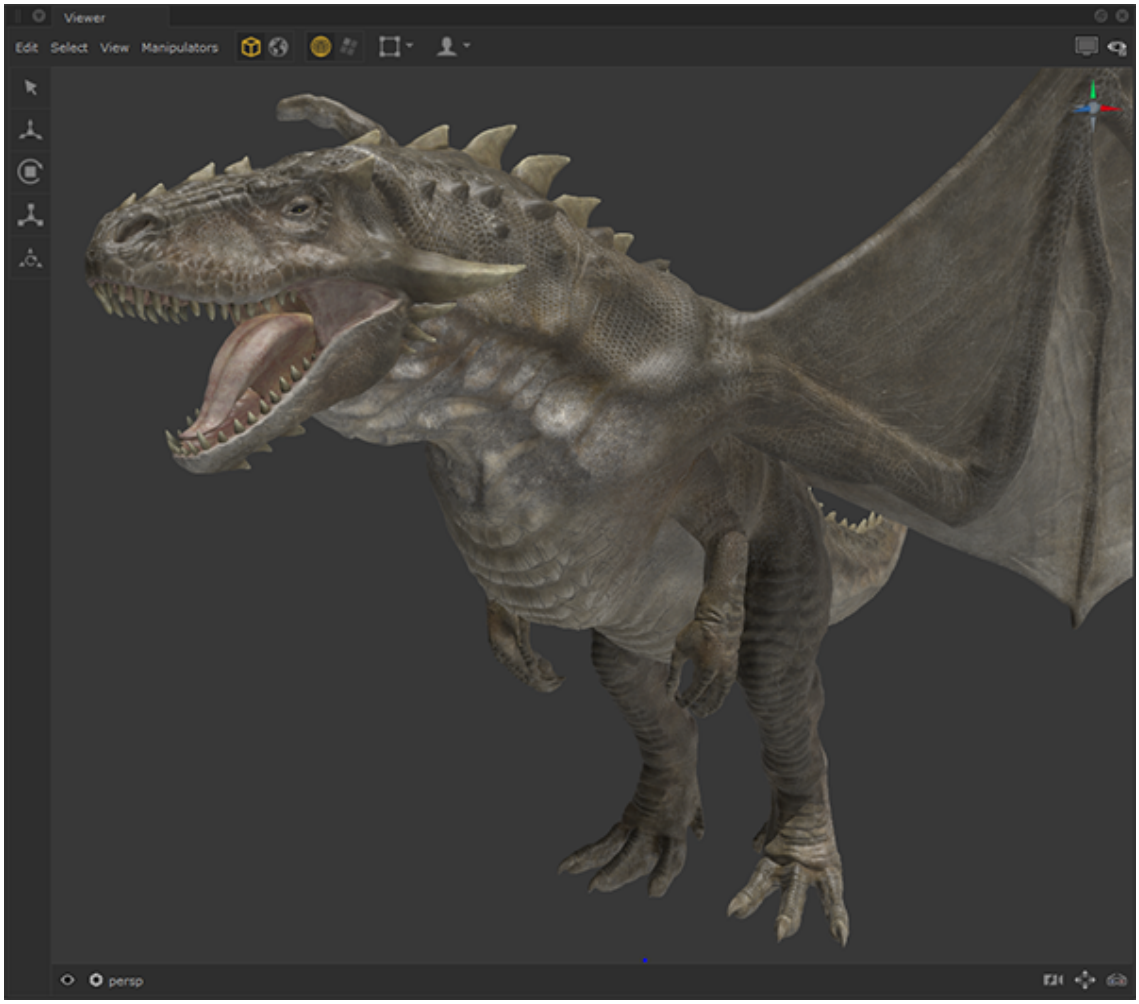
12. Create a **UsdPrimvarReader_float2** node and plug the **result** output into the **st** input on the **UsdUVTexture**.



13. Within the **UsdPrimvarReader_float2**, set the **varname** parameter to **st**.



The texture can now be seen on the object in the Hydra Viewer.

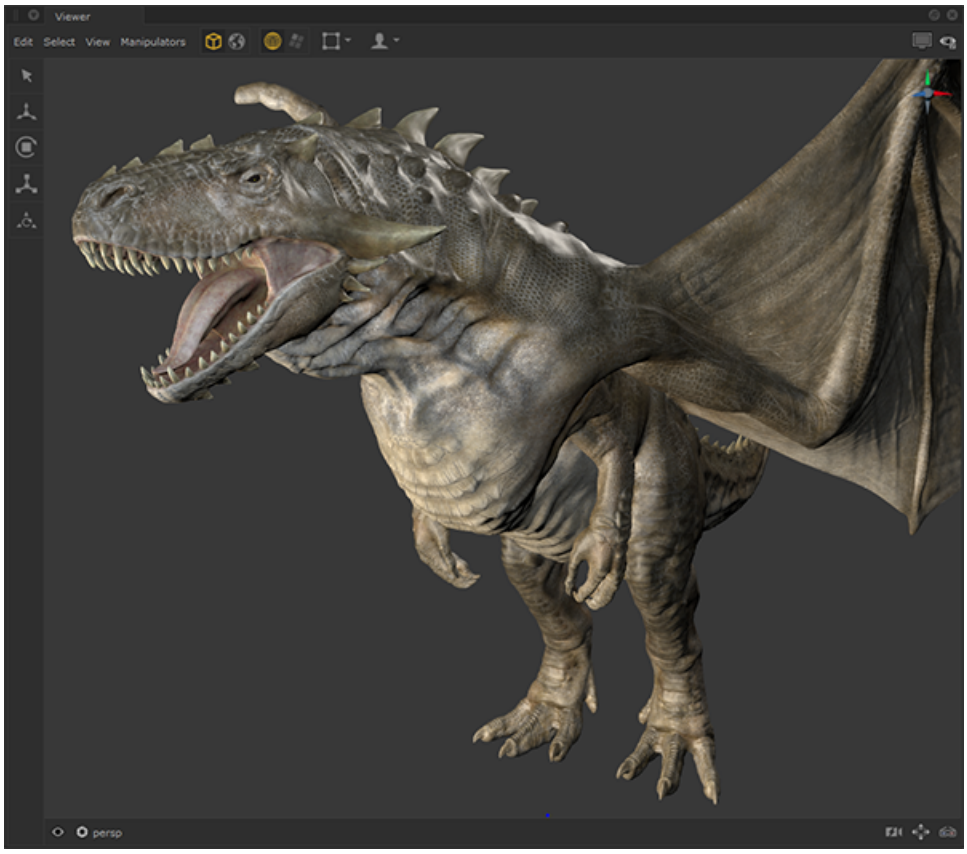


Basic Material off, **Shadows from All Lights**

Video: For a full tutorial series on using USD Preview Surface materials, take a look at our Setting Up USD Preview Surface Materials course.

Setting up USD Lights

As well as setting up USD materials, it is also possible to set up USD lights and view them in the Hydra Viewer.

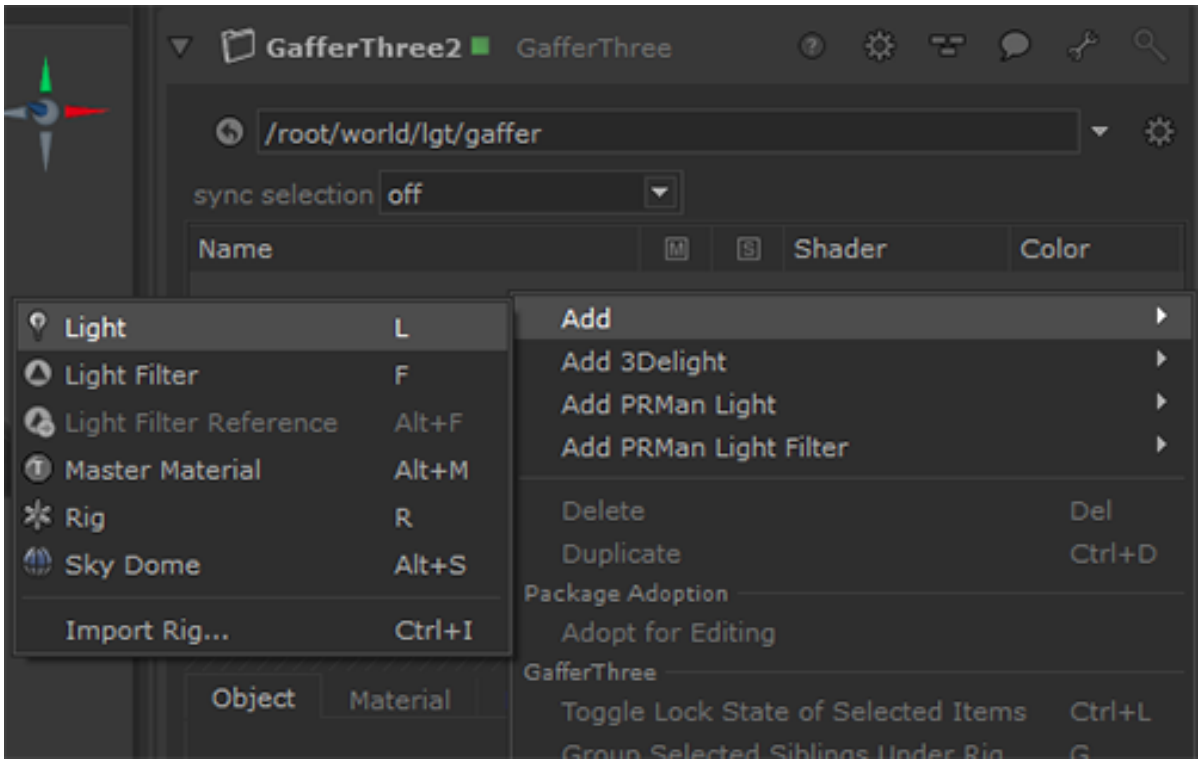


To set up a USD light:

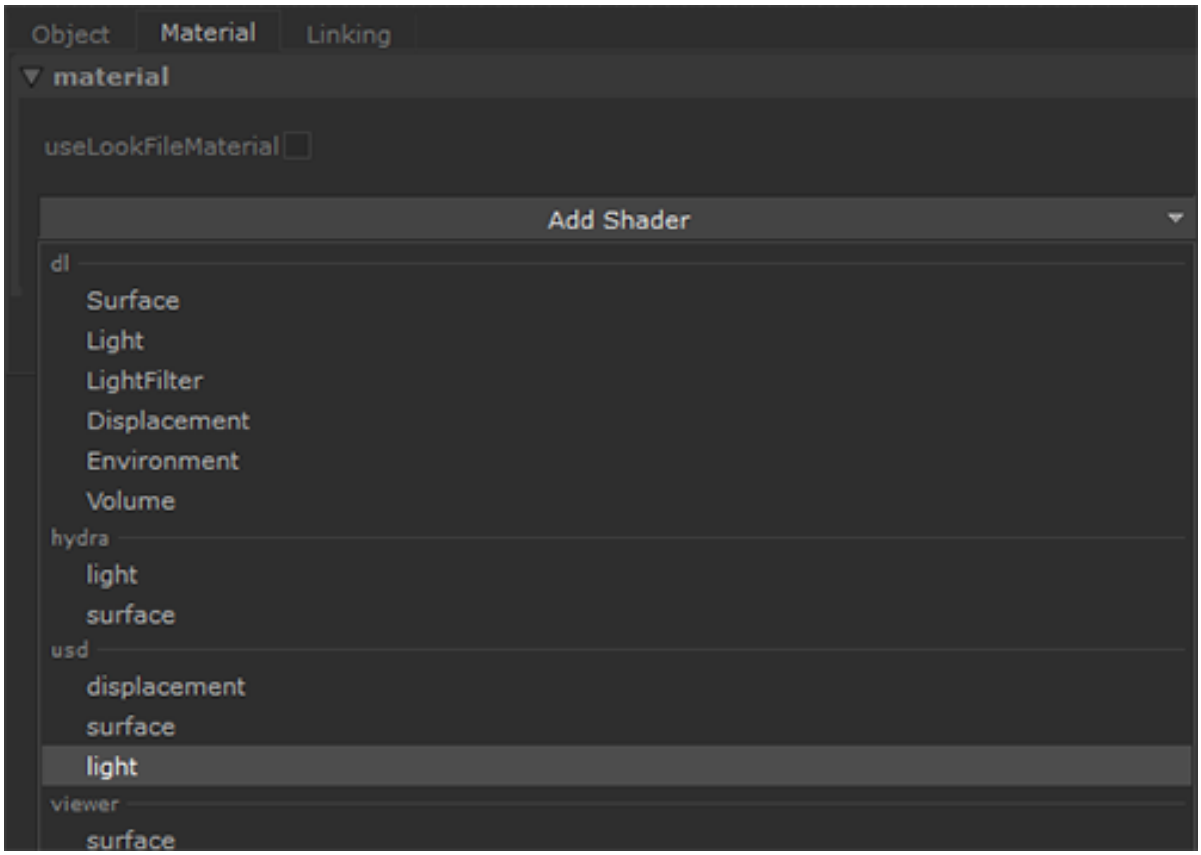
You can use the workflow below to use the **GafferThree** node with a USD Shader.

1. Create a **GafferThree** node and open its **Parameters**, or open the **Parameters** for an existing **GafferThree** node.

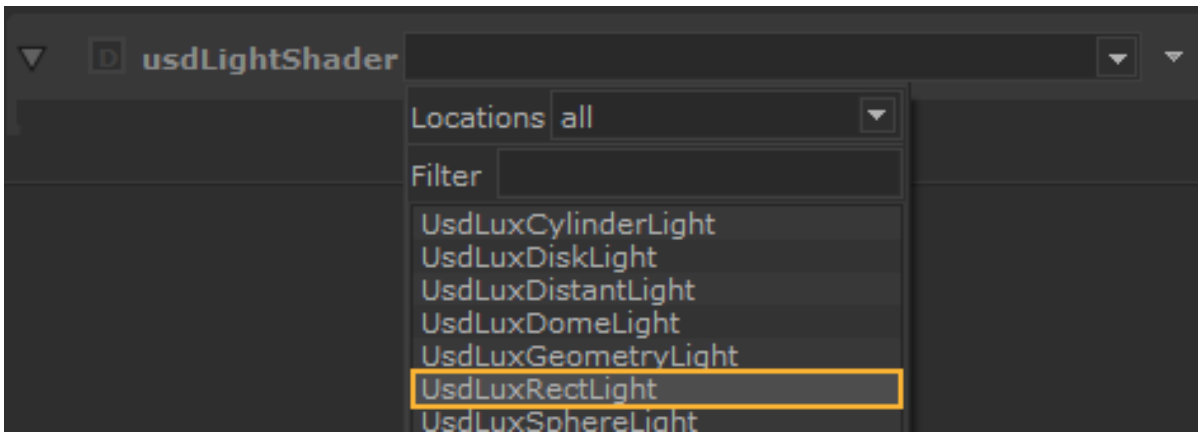
2. In the **GafferThree Parameters** tab, right-click under **Name** and choose **Add** > **Light**.



3. In the **Material** section of the new **light** parameters, click **Add Shader** and choose **usd** > **light**.

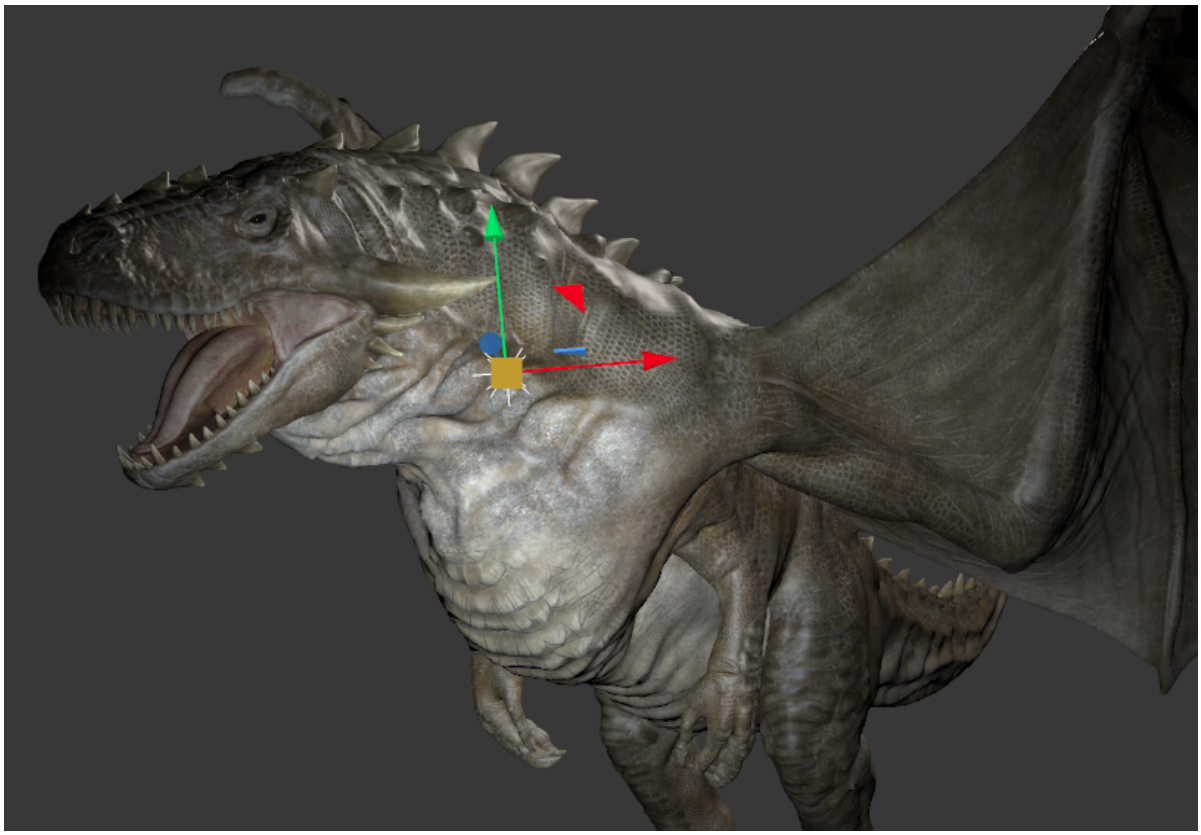


4. In the **usdLightShader** drop-down menu, choose a USD light type, for example, **UsdLuxRectLight**.



The full light material parameters open and you can make adjustments.

5. In the **Viewer** tab, click **View** > **Lighting using USD Lights** to enable the USD lights.
6. Position the light using the **Transform**, **Rotate**, **Scale**, and **Center of Interest** manipulators in the Hydra Viewer.



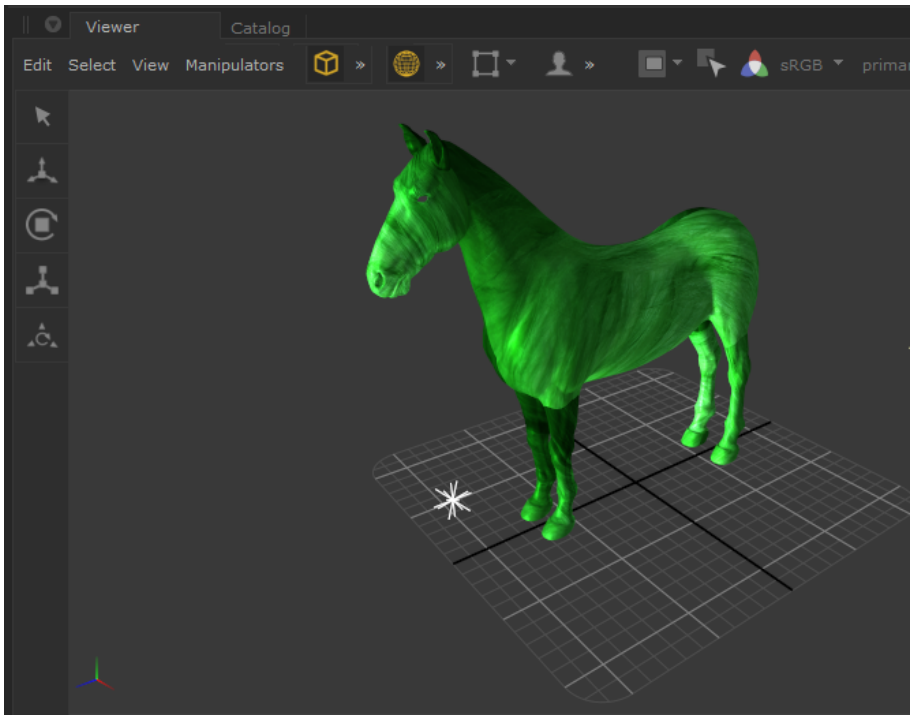
Changing Display Properties for Some Locations

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:

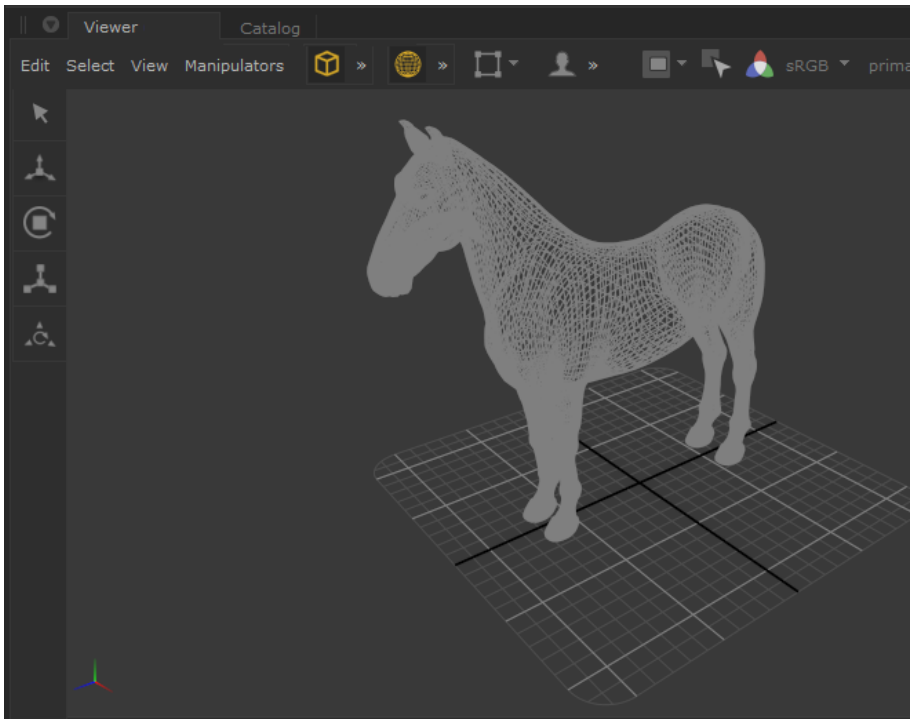
1. Add a ViewerObjectSettings node to the recipe at some point before the current view node.
2. Select the ViewerObjectSettings node and press **Alt**+**E**.
   The ViewerObjectSettings node becomes editable within the **Parameters** tab.
3. Assign the scene graph geometry locations of the objects to 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.
   You can set the following display options 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**).
- **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 an object, with PRMan and Viewer Shader material applied. The Viewer is in **Solid** mode, so the 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.



The following display options can be set in the **annotation** parameter grouping:

- **text** - displays this text with the location in the viewport.
- **color** - the background color of the annotation text.

The following parameters can be set in the ViewerObjectSettings node itself:

- **pickable** - when set to **No**, you can no longer select the object in the **Viewer**.
- **resolveMaterialInViewer** - controls whether the Viewer should always, never, or use default rules to resolve materials.

Customizing the Viewport

Changing the Background Color

You can change the background color 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:

1. Navigate to **Edit** > **Preferences** > **hydraViewer**.
2. Change the **backgroundColor** parameter to the required color values.
   The color changes in the **Viewer** tab.
3. Click **OK** to save changes.

Toggling the Grid Display

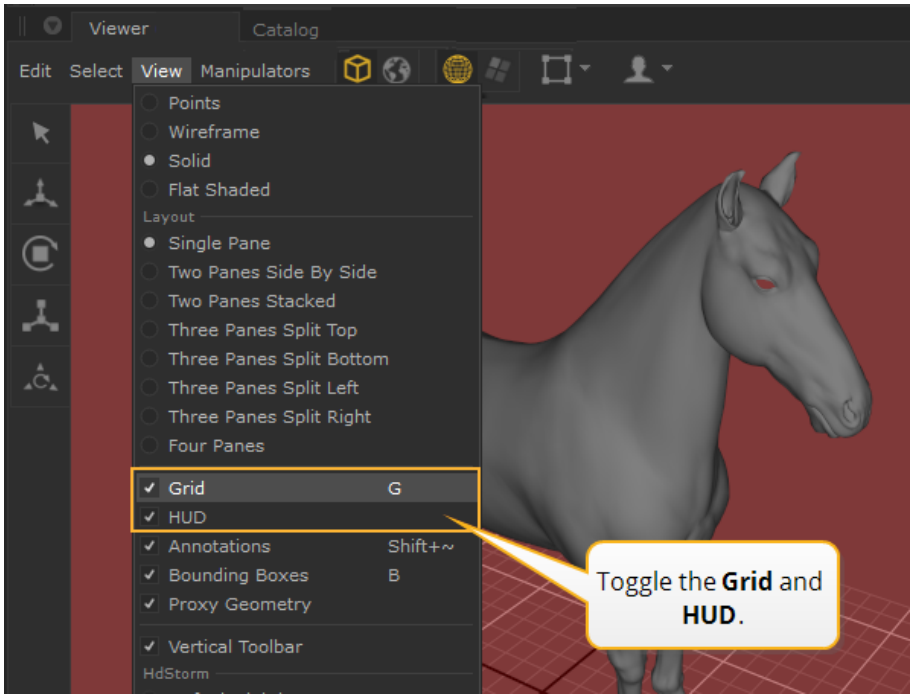
Katana displays a grid to help you get a sense of scale, the origin's location, and the orientation of the XZ plane.

To toggle displaying the grid, in the **Viewer** tab, select **View** > **Grid** (or press **G**).

Toggling the Head-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 cameras have 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 Head-Up Display (HUD), select **View** > **HUD** (or press **H**).



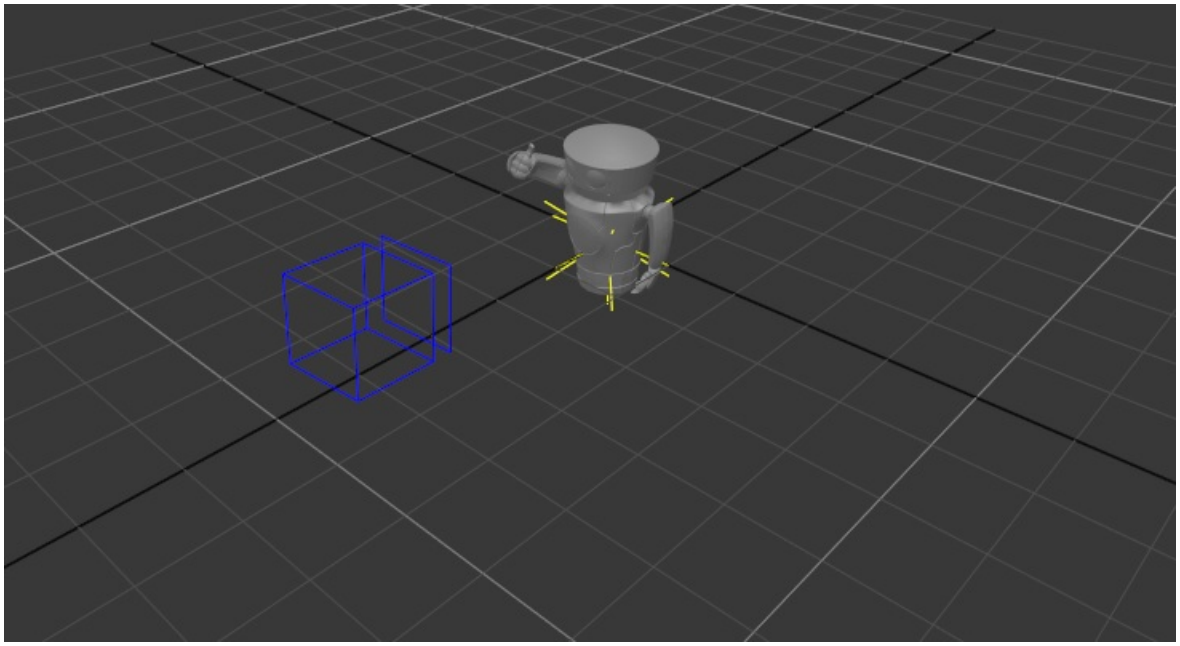
Lighting Your Scene

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.

Katana does not assume a given scene scale. There are no physics solvers, like those in Maya, that need to know what real world measurement a unit represents. If you're using a shader library that has real world units for shade parameters, such as emissive lights with power per unit area, that's handled by the shader implementation rather than Katana.

The unit of measure used is up to you, because Katana equates any value as being equal to one unit in 3D space. The image shows a simple scene containing an object and a camera. If the large white square in the Viewer represents one unit of measure, so the smaller squares represent one tenth of that unit. So, if you measured on set in meters, one small square could be equal to a meter, centimeter, or millimeter.



Creating a Light



Note: Lights can also be created using **Lighting Tools** within the **Viewer**. For more information on this advanced workflow, see Lighting Tools.

Creating a light inside Katana can be done in several ways:

- using separate nodes (LightCreate and Material) creating it from its core components, or
- using the GafferThree node, which packages up light creation with a number of other useful functions, or
- using the UsdLight node to create a native USD light

To create a light in Katana 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** > **dl**> **light** within the Material node's **Parameters** tab.

   A new 3Delight light shader is added to the Material node.



Note: The shader you select in the **Add Shader** dropdown doesn't necessarily need to be **dl**. 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 **dl** shader has been chosen.

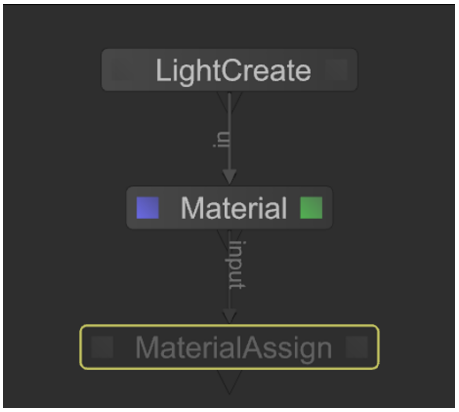
5. Click the arrow to the right of **dlLightShader** to display the shader options.



6. Select the type **areaLight** from the dropdown.
   The light name is populated in the dropdown button.



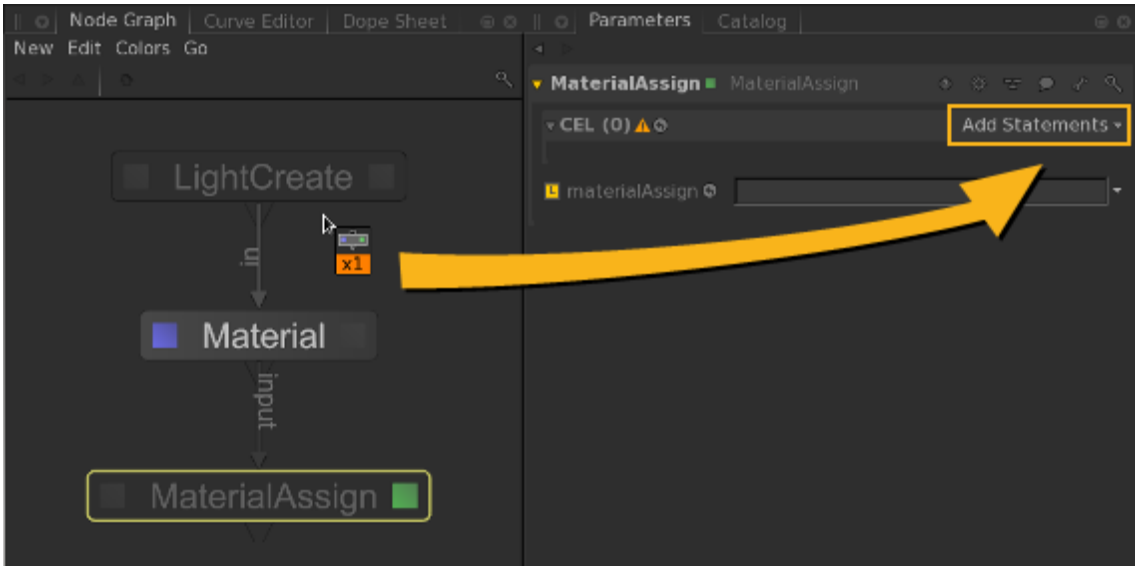
7. Create a MaterialAssign node and connect it to the output of the Material node.



8. Select the MaterialAssign node and press **Alt**+**E**.
   The MaterialAssign node becomes editable within the **Parameters** tab.

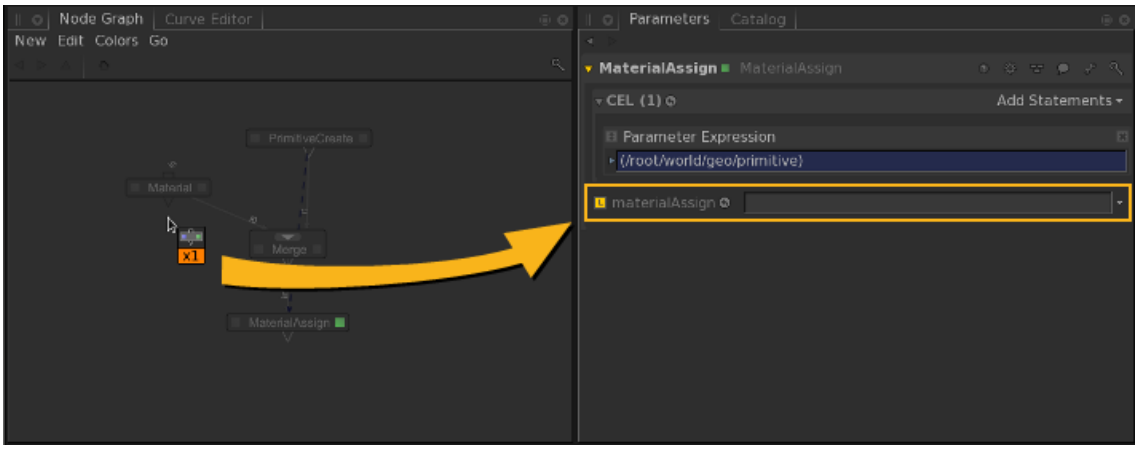
9. **Shift**+middle-click and drag from the LightCreate node in the **Node Graph** tab to the **Add Statements** dropdown in the **Parameters** tab.

   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 3Delight light shader defined in the Material node is now assigned to the light defined in the LightCreate node.

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:

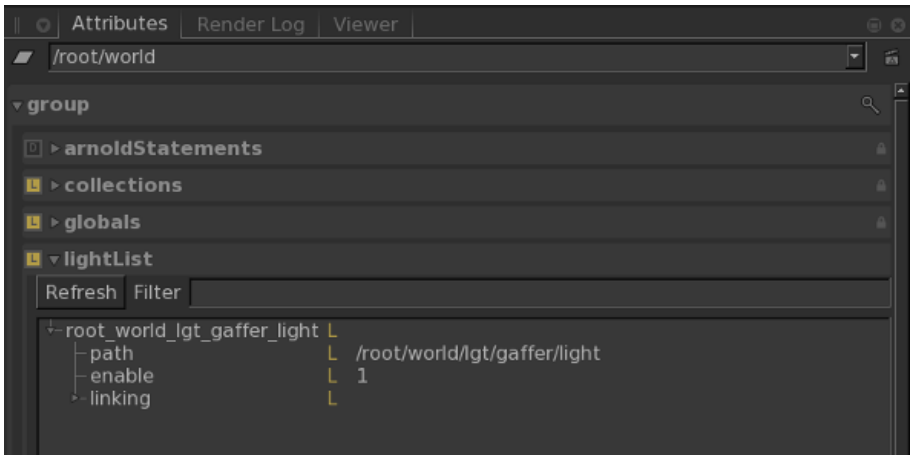
- Translate and rotate the light with the manipulators, (see Transforming an Object in the Viewer), or
- Look through the light and change its view position (see Changing What You Look Through).

Light Linking

Light linking is a typical example of how a standard setting is defined high up in the hierarchy and then overridden at a specified scene graph location where a different setting is needed. Shadow Linking works in the exact same way.

1. In an empty scene, create a sphere using a PrimitiveCreate Node.
   Set the name to **/root/world/geo/sphere**.
2. Add a plane with a PrimitiveCreate node.
   Set the name to **/root/world/geo/plane**.
3. Add a Merge node and connect both PrimitiveCreate nodes as inputs.
4. Add a light with a GafferThree node.
5. Connect the output of the Merge node to the input of the GafferThree node.
6. Add a LightLink node.
7. Connect the output of the GafferThree node to the input of the LightLink node.
8. Select the LightLink node and press **Alt**+**E** to edit it.
9. Set the **effect** field to **illumination**, and the **action** field to **off**.
10. Add the primitive sphere to the objects field.
11. Add the light to the lights field.

Creating the light adds a **lightList** attribute group under **/root/world** with the **enable** attribute set to 1.



Note: With the default behavior of the light set to off - by changing the **defaultLink** dropdown to **off** - the enable attribute is set to 0.

The primitive sphere's **lightList** enable attribute is set to 0, as it is overridden locally by the LightLink node. Attributes are inherited from parent scene graph locations. If needed, they can be locally overridden as shown above where a specific light (**/root/world/lgt/gaffer/light1**) is disabled for a certain node (**/root/world/geo/sphere**).

Note: A **G** label next to an attribute signifies that its value has been inherited from a parent scene graph location, whereas the label **L** means the attribute is stored locally at the selected location.

Getting to Grips with the GafferThree Node

The method described in Creating a Light, 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.
- Add light filters and light filter references.
- Mute and solo lights and rigs.
- Link lights to specific objects.

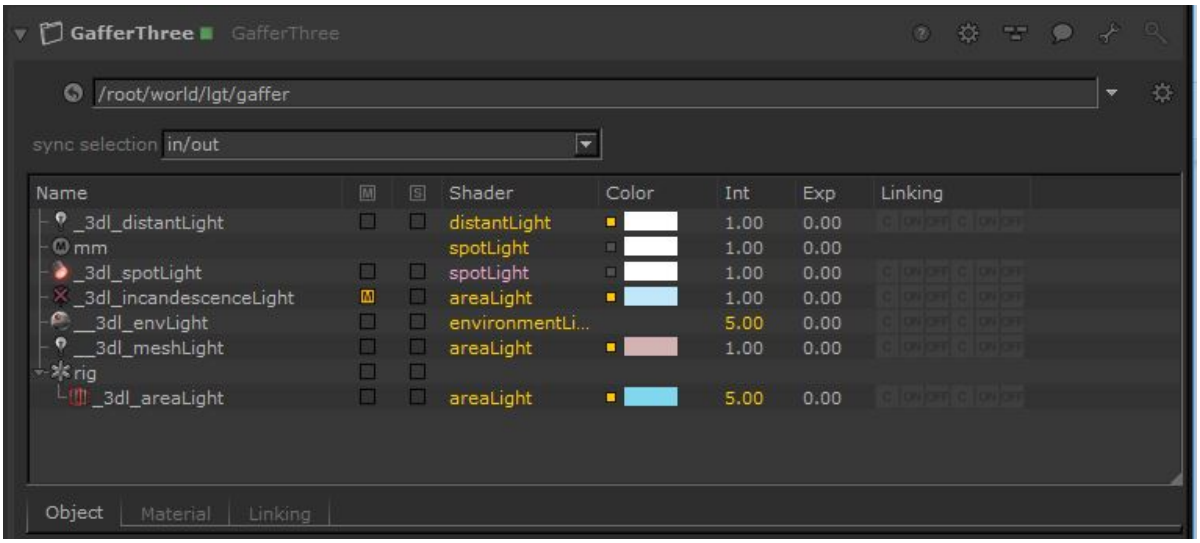
• Add aim constraints to lights.

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, light filters, light filter references, and Template 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 Template 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 Using and Overriding Look Files with GafferThree Lights

Creating a Light Using the GafferThree Node



Note: Lights can now be created using **Lighting Tools** within the **Hydra Viewer**. For more information on this advanced workflow, see Lighting Tools.

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:

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 Template 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.

You can add a light shader to a light, either through 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 **Spotlight** from the list.

   The name of the light shader, **Spotlight**, appears in the **Shader** column in the Gaffer object table and in the **Material** tab as well.

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 **dl Light** shader from the list.

   The renderer-specific shader appears in the **Material** tab.
5. Click on the newly added shader and select **Spotlight** from the list.

   The name of the light shader, **Spotlight**, appears in the **Material** sub-tab and in the **Shader** column of the Gaffer object table as well.



Note: 3Delight lights, with shaders pre-applied are available to add to the Gaffer table via the **Add** menu or the associated keyboard shortcuts.

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, dl (3Delight) 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 Template Light Material

At times it is best to have a Template Material and set local overrides per light. This can be done within the GafferThree node by creating a Template Material and assigning it to a light. Any changes made within the light's **Material** sub-tab act as an override for the Template Material.

Creating a Template Material

In the GafferThree node's **Parameters** tab , right-click in the Gaffer object table, and select **Add** > **Template Material** or press **Alt**+**M**.

A Template Material location is created inside the GafferThree node.

Assigning a Template 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 Template Material to.
2. Select the Template Material from the list (Template Materials are displayed in pink).

   The Template 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.

To add an aim constraint to a light:

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).

To change 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 a Light Filter Using the GafferThree Node

Light filters allow you to modify a light's behavior in order to give these light sources additional effects. Light filters are renderer-specific, but regardless of renderer, you can add them in the GafferThree node in the same way.

To create a light filter with the GafferThree node:

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 Filter** or press **F**.

   A light filter is added in the Gaffer object table.



Note: The light filter location is created under **/root/world/lgt/gaffer** by default. You can change the scene graph location by setting a new path in the root location field in the GafferThree node's **Parameters** tab.

Creating a Light Filter Reference Using the GafferThree Node

Light filter references are light filters attached to a light or skydome that reference another light filter package. These packages allow you to apply multiple light filter effects as only one light filter on a pre-existing light.

To create a light filter reference with the GafferThree node:

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.
4. Right-click the light and select **Add** > **Light Filter Reference** or press **Alt**+**F** when the light is selected.

   A light filter reference is added to the light.
5. Select the light filter reference and click on the **Object** tab beneath the object table list. Specify the path to the light filter package in the **referencePath** field.

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 or light filter 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 or light filter 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 or light filter, follow the steps below:

1. In the Gaffer object table's **Parameters** tab, select the light or light filter 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 or light filter 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 GafferThree.

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 an Incoming Scene

You can adopt lights, rigs, and Template Materials from any upstream GafferThree nodes and then override any of their set parameters.

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 Template 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 Template 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 Template 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, Light Filters, 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 light filters and rigs works the same way.

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: To solo a light filter or a rig, follow the same procedure but select a light filter or rig instead of a light.

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 light filters or rigs, follow the same procedure but select the light filters or rigs you want to mute, instead of a light.

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.

Using and Overriding Look Files with GafferThree Lights

To use more complex shading networks to drive your light's materials, you can define the material first, and then use a look file to apply it to lights created in GafferThree nodes.

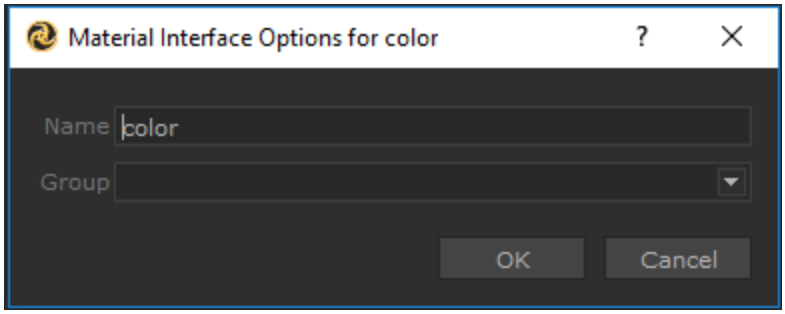
To set up the network material, and bake it into a look file:

1. Add a NetworkMaterial node. In it's parameters tab, click add terminal, and choose **dl/Light** from the dropdown
2. Add a DlShadingNode and change the **nodeType** to **spotLight**

Displaying Network Material Parameter Values

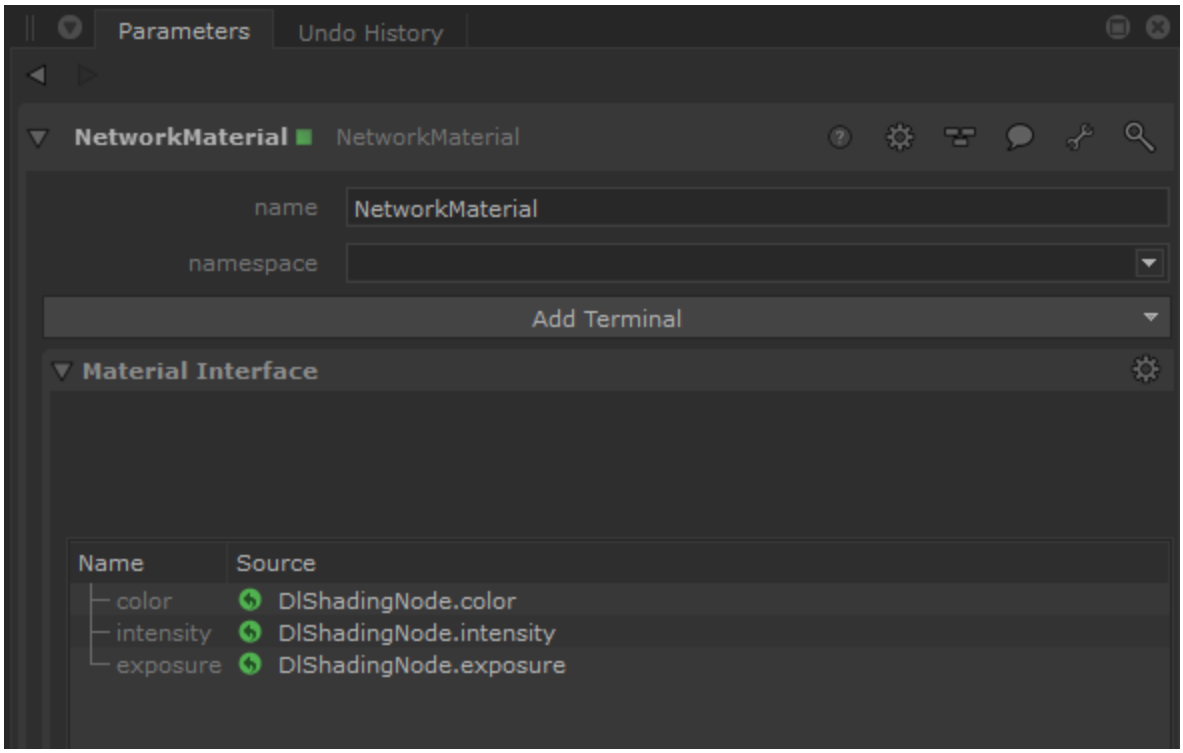
To show values of parameters from the public interface of the Network Material in specific columns of the GafferThree object table, you need to set parameter meta name attributes in the **material** group attribute of the **material** location that defines the Network Material. To set these attributes, do the following:

1. In the DlShadingNode, click the wrench icon to the right of the Color controls, select **Edit Parameter Name in Material Interface...**, and change the **Name** value to **color.**



2. Using the wrench button for both **Intensity** and **Exposure**, change **Name** parameter to **intensity** and **exposure** respectively.
3. Change the **Color**, **Intensity**, and **Exposure** values to non-defaults.

   The parameters are displayed in the **Material Interface** section of the **NetworkMaterial** node's parameters tab.



Note: The naming of the parameters, and their related attributes, need to follow a strict syntax to be read as metadata by the gafferThree table.
Attributes must be set to read the material parameters and feed their values as metadata to the gaffer object table. These attributes need to adhere to the following syntax:

Names of the attribute: `material.meta.[parameterMetaName].[rendererName]`
For example, **material.meta.exposure.dl**
The values: `parameters.[parameterName]`
For example, **parameters.exposure**

You can set these string attributes one at a time using AttributeSet nodes, or you can set multiple attributes at once using OpScript nodes.

4. Add an OpScript node and use the following script, which will populate the material.meta attributes with the names and values of the parameters declared in the network material.

```
local materialInterfaceGroupAttr = Interface.GetAttr("material.interface")

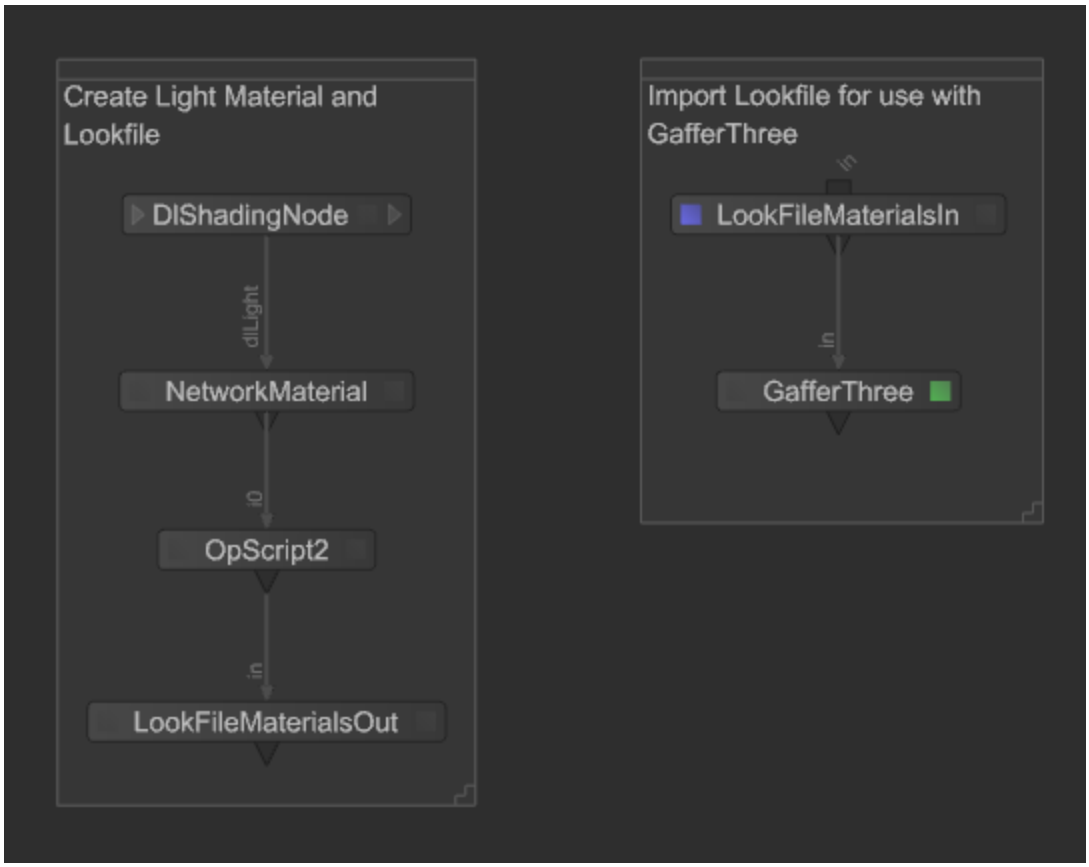
local targetName = "dl"

for i = 0, materialInterfaceGroupAttr:getNumberOfChildren() - 1 do
    local parameterName = materialInterfaceGroupAttr:getChildName(i)
    Interface.SetAttr("material.meta." .. parameterName .. "." ..
```

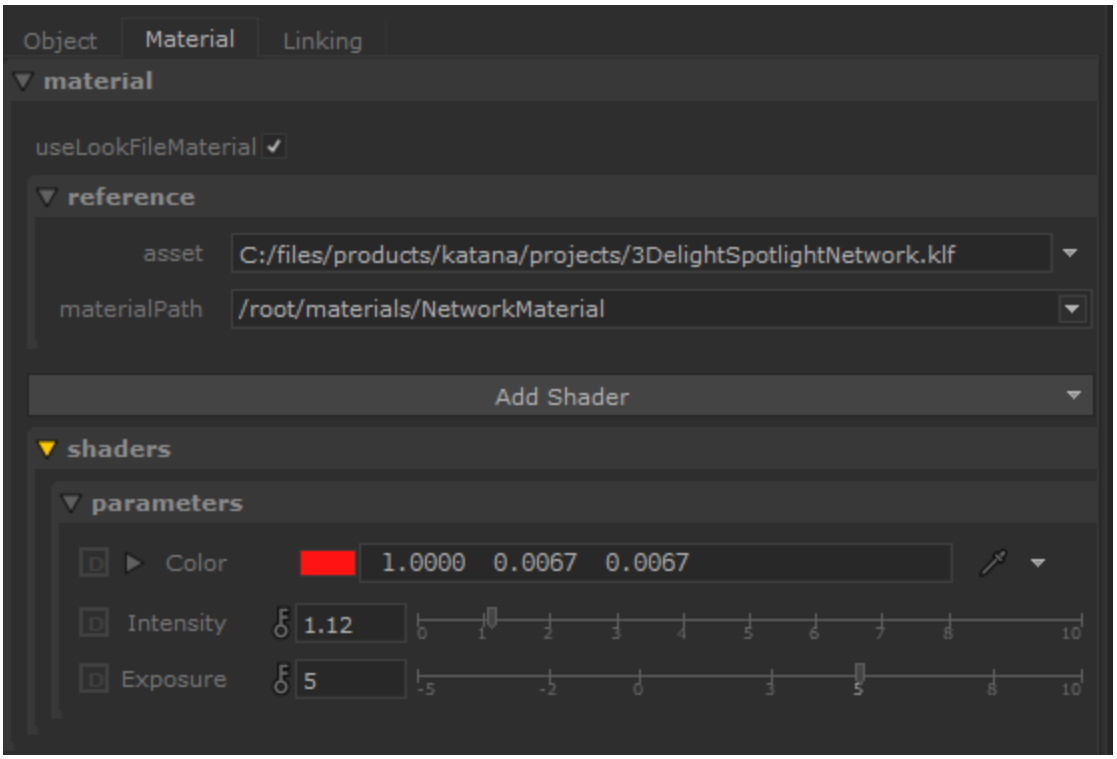
```
targetName,
                        StringAttribute("parameters." .. parameterName))
end
```

5. Bake a material lookfile, by using a **LookFileMaterialsOut** node. Set the **saveTo** location and click **Write Look File** and save the look file.
6. In a separate branch of the NodeGraph, add a LookFileMaterialsIn node. In the **lookfile** section, browse to your saved lookfile from step 5.
7. Add a gafferThree node.

   Your node graph should now look similar to this:

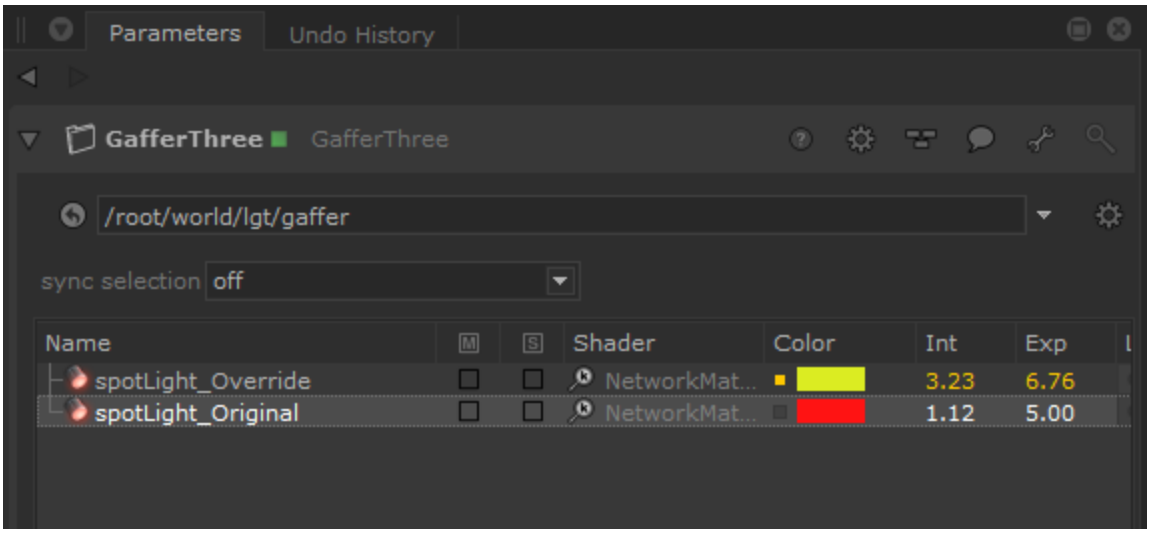


8. Add a 3Delight spotlight by right-clicking in the gaffer table, select **Add3Delight** > **Spotlight**, or type **Q**.
9. Select the spotlight in the gaffer table, open the **Material** tab below, check **useLookFileMaterial**.
10. In the **asset** field, browse to the look file that was baked in step 5.
11. Middle-mouse and drag the network material that appears in the **Scene Graph** tab under **root/materials/NetworkMaterial**) into the **materialPath** field.



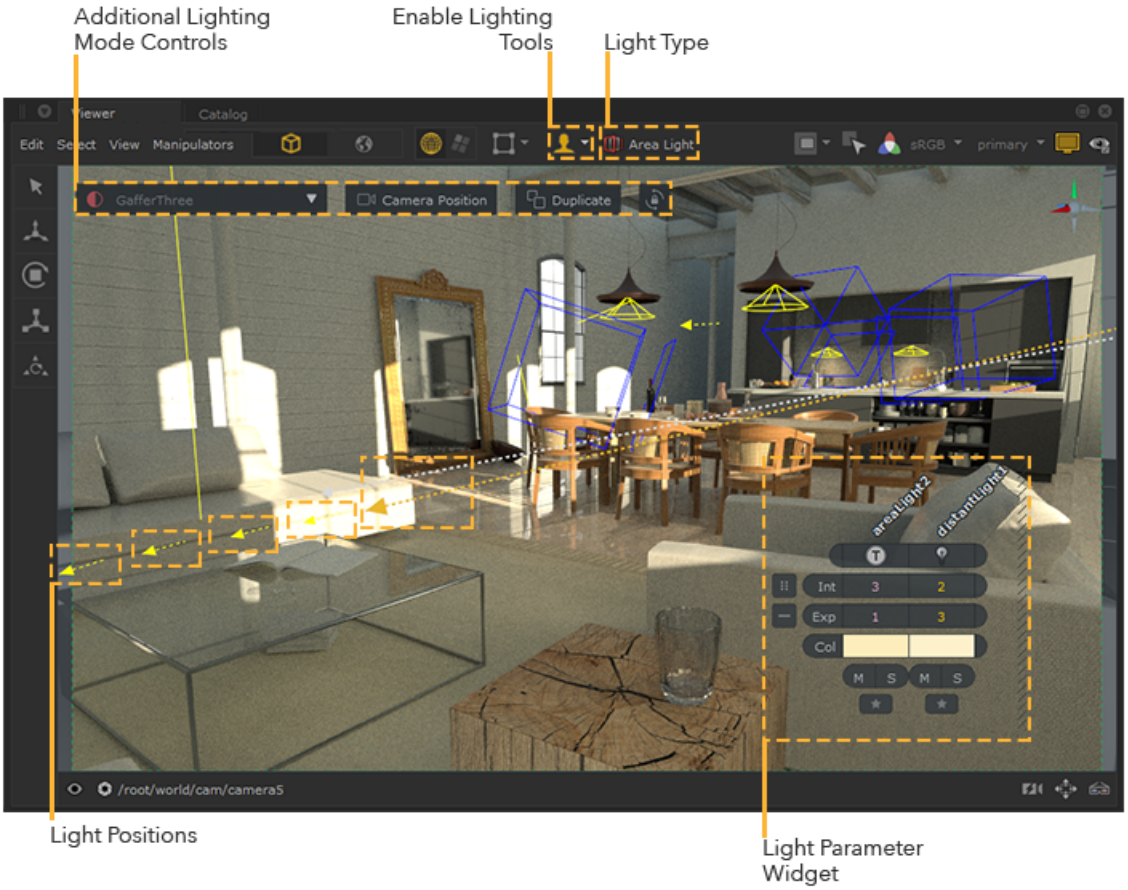
The exposed parameters from your network material will be listed as editable parameters. These parameters and their values will also be displayed correctly in the **Color**, **Int** and **Exp** columns of the gaffer table for ease of reference. Any overrides to the material parameters are reflected here.

The same look file can be applied to many different lights, and locally overridden in the GafferThree node.



Lighting Tools

Katana's Lighting Tools enhance lighting workflows for artists. When the **Lighting Tools** button is enabled, artists can work full-screen in the **Viewer**, streamlining the creation and editing of lights in your scene. This workflow is inspired by the thought processes of live action cinematographers and lighting artists, allowing you to work quickly and smoothly with more creative freedom.



Lighting Tools UI

Katana's Lighting Tools reduce the number of steps required to place lights and increases the accuracy of the placement. This intuitive way of working embraces the way lighting artists think so you can focus on the result, not the process.

Creating and Placing Lights

Find out how to create and place lights using Lighting Tools in the Hydra Viewer.

Editing Lights in the Parameter Widget

Discover how to edit your lights and customize the parameters shown on your light parameters widgets.

Clones and Template Materials

Learn how to further streamline your lighting workflows using cloning and Template Materials.

Creating Lights using Lighting Tools

Katana's Lighting Tools introduce an artist-friendly environment to set up lights in a **GafferThree** node within your scene. It allows you to interact directly with your image during a live render session.

Lighting Tools are especially useful when artists want to work in a purely creative way without being slowed down or distracted by unnecessary panels.

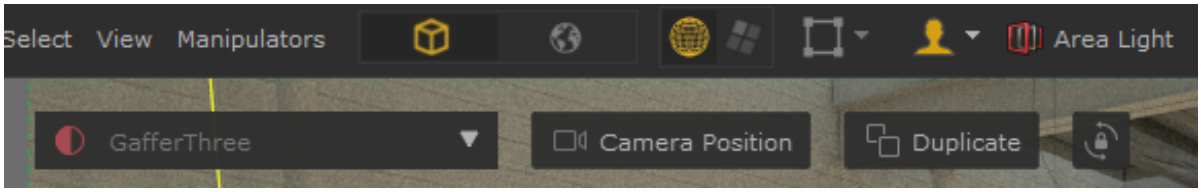
How to Place a Light using Lighting Tools

1. Create a **GafferThree** node and place it in your **Node Graph**.

   Note: For more information on the **GafferThree** node, see Getting to Grips with the GafferThree Node.

2. Click the **Lighting Tools** button  in the **Viewer** tab, or press **L** on the keyboard to turn on Lighting Tools.

   Additional buttons are displayed in the Hydra Viewer.



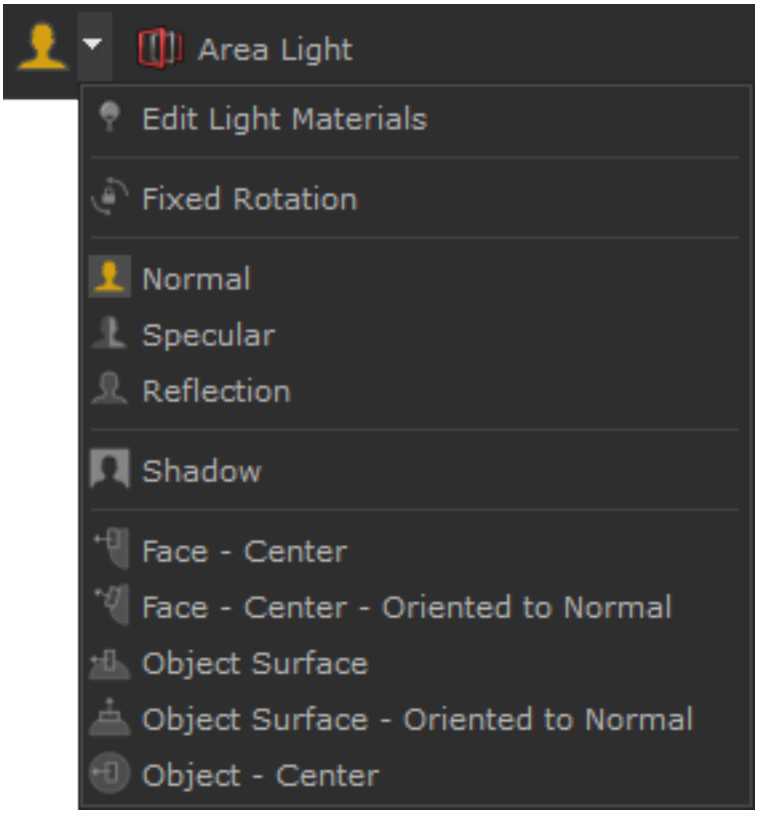
   If you have more than one **GafferThree** node in your scene, you can select which **GafferThree** node you would like to use to create your light by clicking on the **GafferThree** dropdown.



You can no longer select any geometry in the scene.

Tip: Press **Spacebar** over the Hydra Viewer to enter full-screen mode and work directly on your image.

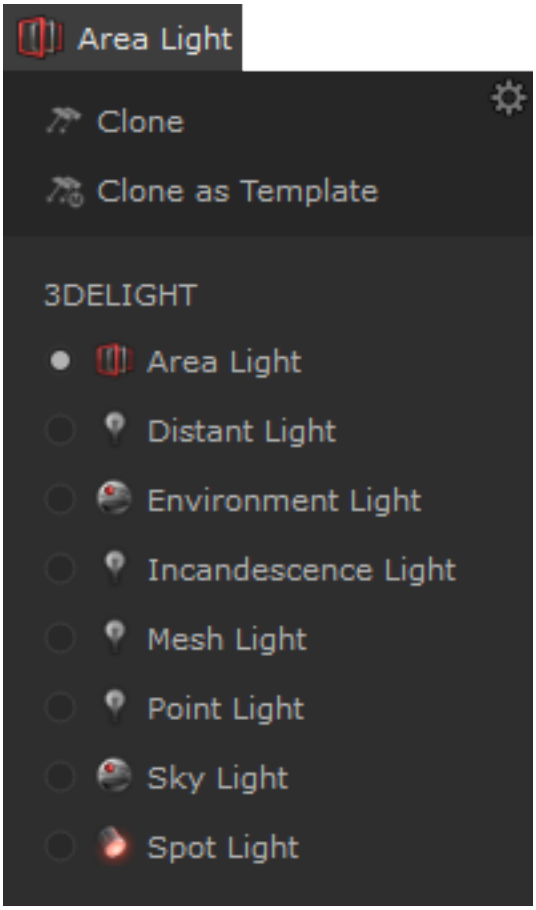
3. Click the dropdown arrow next to the **Lighting Tools** button to choose a Lighting Mode. You can also hold **Shift** and press **L** to cycle Lighting Modes.





Note: For information about each Lighting Mode, see the Lighting Modes section in this topic.

4. Choose the light you want to create from the **Lights** dropdown menu.



Your default renderer's lights appear in the list. If you want to choose a light from a different renderer you have set up in Katana, click the settings icon  from within the **Lights** dropdown menu, and select the renderers you want to access from the dropdown menu.



Note: For information about the **Clone** and **Clone as Template** options, see Cloning Lights and Template Materials.

5. **Shift** + **Click** to place a light.

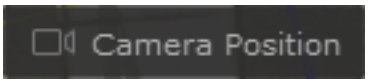
A light parameter widget is displayed.



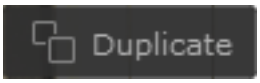
Note: For information on editing lights using the light parameter widget, see the Editing Lights using Lighting Tools section of this topic.

You can also place lights at the same position as the camera you are currently looking through, and other lights in you scene.

To place a light at the same position as the camera you are looking through, click the **Camera Position** button.



To place a light at the same position as an existing light in your scene, select the light and press the **Duplicate** button.

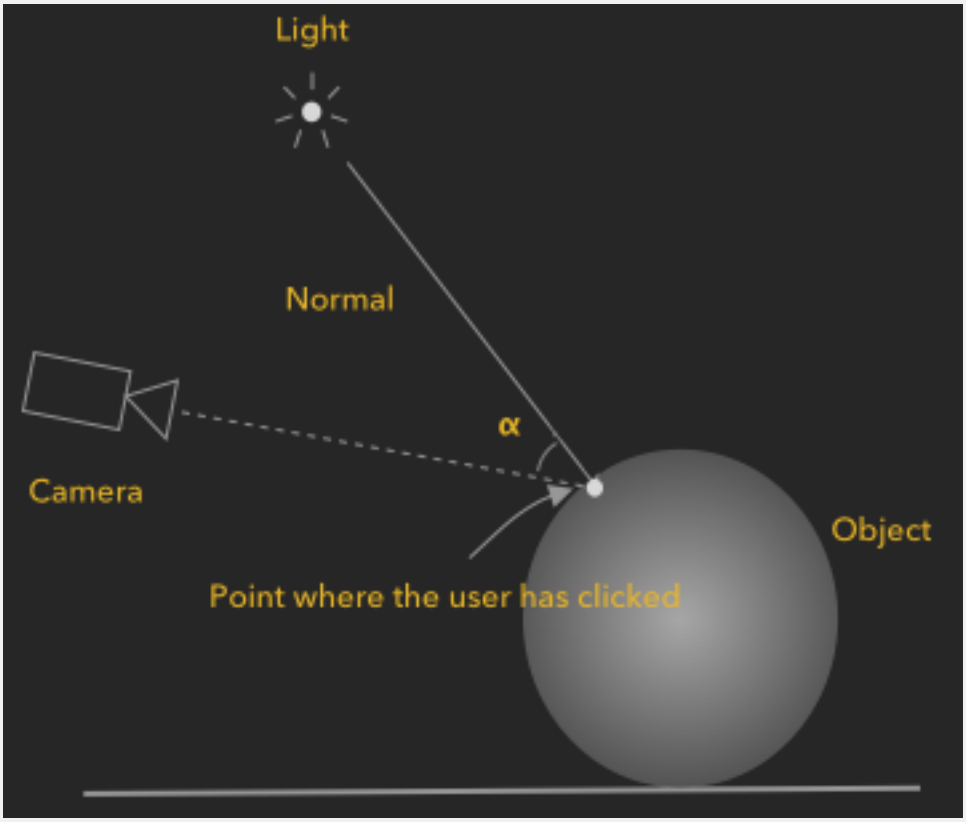


Note: It is also possible to clone lights using the **Clone** and **Clone as Template** options in the **Lights** dropdown menu.
For more information, see Cloning Lights and Template Materials.

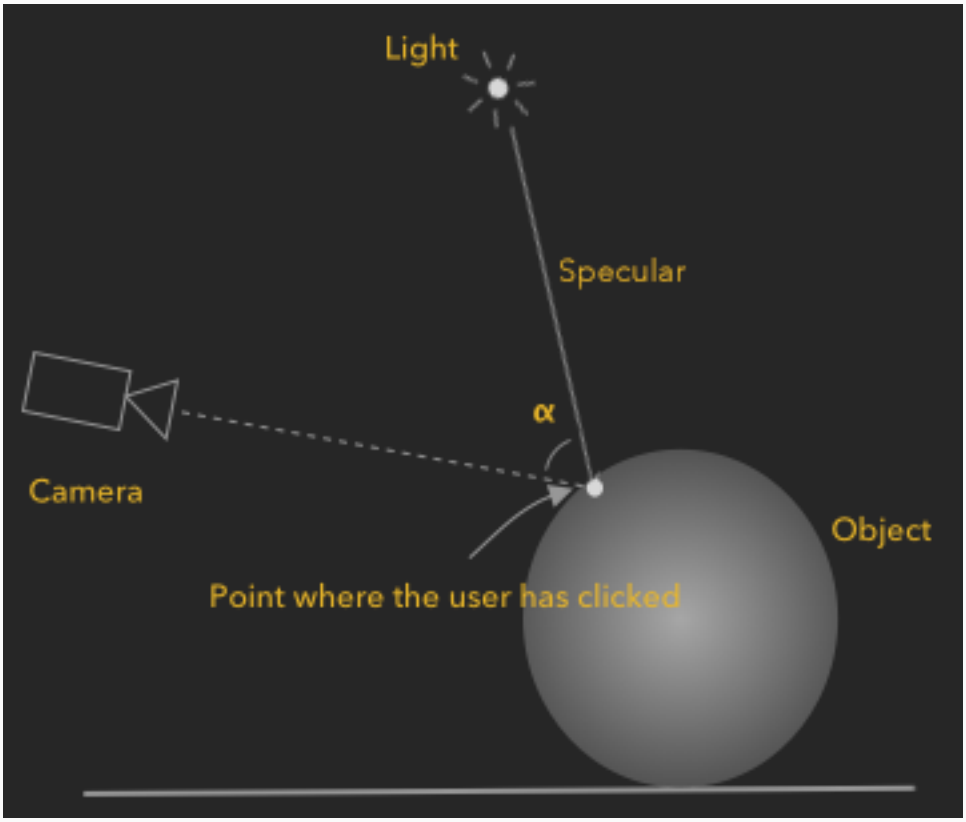
Lighting Modes

The following lighting modes are available from the Lighting Tools dropdown menu:

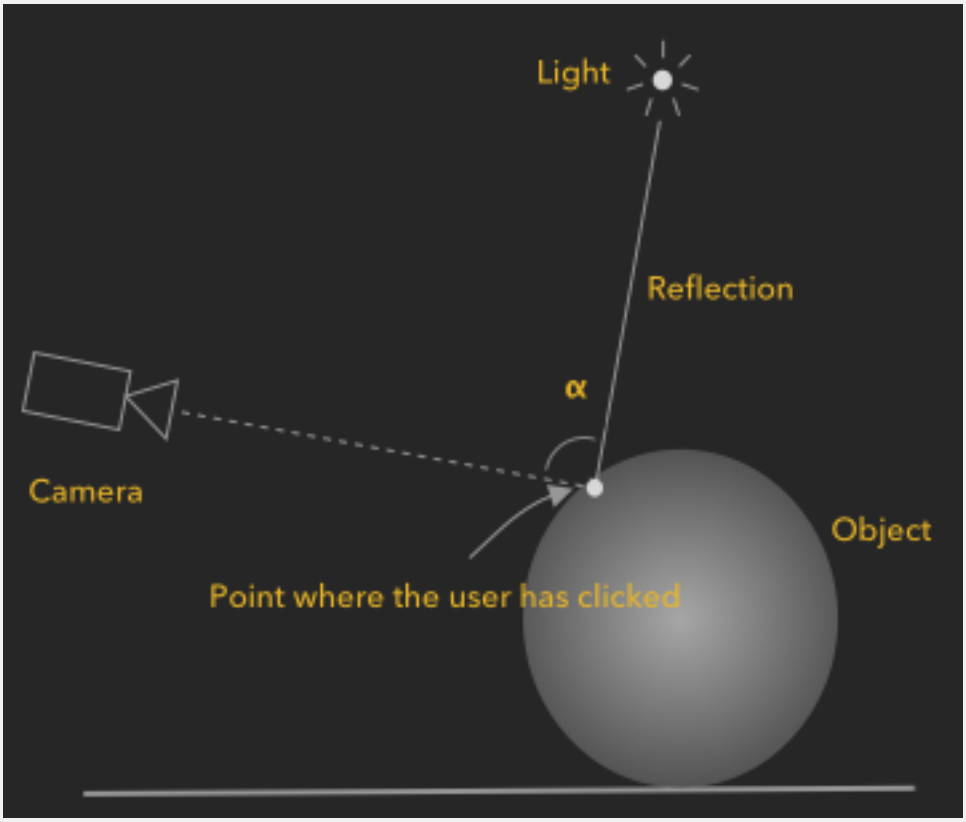
 **Edit Light Materials**	When enabled, new lights cannot be created and existing lights cannot be moved. You can only select lights to open their light parameter widget and make edits to their parameters. This mode is useful if you have finalized the light positions in your scene and want to make sure you cannot move them accidentally while adjusting the parameters.
 **Fixed Rotation**	Make small adjustments to a light's position without changing the angle of the light. This mode is useful when you have finalized the angle of light you want but want to change the position slightly.  **Note:** For more information, see the Fixed Rotation section of this topic.
 **Normal**	The direction of the light is oriented to the normal of the geometry. This mode places the light directly above the point of placement on the geometry and gives a diffuse appearance.



/ **Specular**	The direction of light is exactly halfway between the normal and the reflection angles. This mode is useful if you want a soft transition between a diffuse light and a sharp reflection.



/ **Reflection**	The direction of the light is oriented to the reflection of the object's surface. This mode is useful for placing a specific light reflection precisely where you want it to appear on your geometry.



Shadow

When enabled, you can create a light as normal and also create a shadow point beyond the light which can be positioned.

This mode give you full control over where shadows are cast.

Note: For a workflow example using Shadow mode, see the Shadow Mode Workflow Example section in this topic.

Face - Center

These modes come from the **Snapping** toolset, they allow you to snap a light to a target location's Faces, Center, or Object.

These modes can be useful for placing a light in a specific location on a piece of geometry. For example, if you would like to place a light in the very center of a light bulb object, you can use **Object Center** mode to ensure the light is placed precisely in the correct position.

/ **Face - Center - Oriented to Normal**	
/ **Object Surface**	
/ **Object Surface - Oriented to Normal**	
/ **Object Center**	

Note: For more information about **Snapping** modes, see Snapping Modes.

How to Position Lights Using Lighting Tools

There are a few different ways of positioning lights using Lighting Tools:

- **Click** + **Drag** a light's arrowhead to move it along the surface of the object.



- **Ctrl** + **Click** + **Drag** to move the light towards and away from the object.

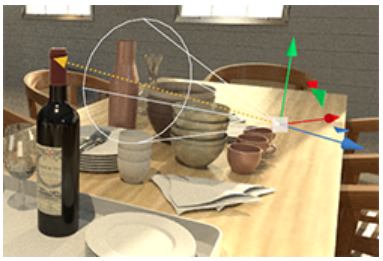


- **Ctrl** + **Shift** + **Click** + **Drag** to scale the light uniformly.



- Use the manipulators in the toolbar to the left of the **Viewer** tab to Translate, Rotate, and Scale the light.

  You can also use the Center of Interest manipulator to position your lights.



Translate light



Rotate light



Scale light

If you want to make small adjustments to a light's position without changing the angle of the light, you can use **Fixed Rotation** mode. **Fixed Rotation** mode allows you to move a light by snapping the Center of Interest head to the surface of the geometry, without rotating the light.

1. Click on the **Fixed Rotation** button in the Hydra Viewer or press **O** on the keyboard to activate **Fixed Rotation** mode. You can also select **Fixed Rotation** mode from the **Lighting Mode** dropdown menu.

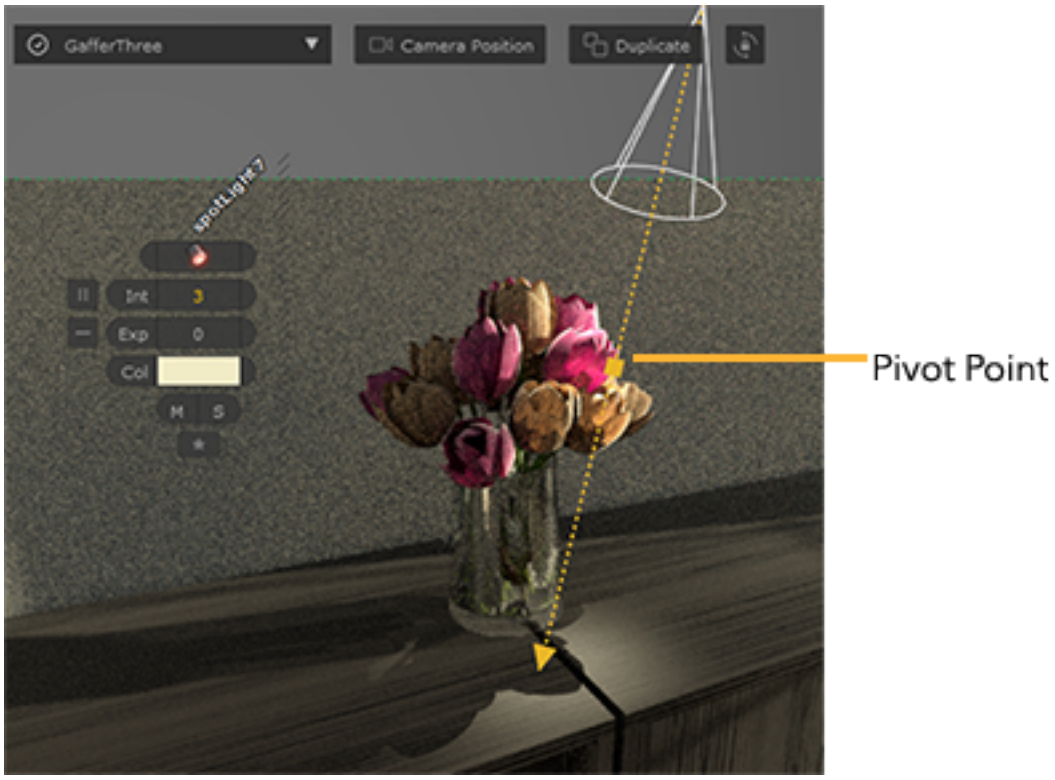
   Note: Press and hold **O** to temporarily activate **Fixed Rotation** mode.

2. Click or marquee select the light(s) you want to move.
3. **Click** +**Drag** the light(s) into position.

Shadow Mode Workflow Example

Shadow mode allows you to place a light while taking into account where you would like the shadow from the light to be cast in your scene. This is useful if you want a shadow from a particular object to be cast in a specific location. **Shadow** mode is a fast and accurate way of placing lights and provides you with a high level of control in situations where shadow placement is important.

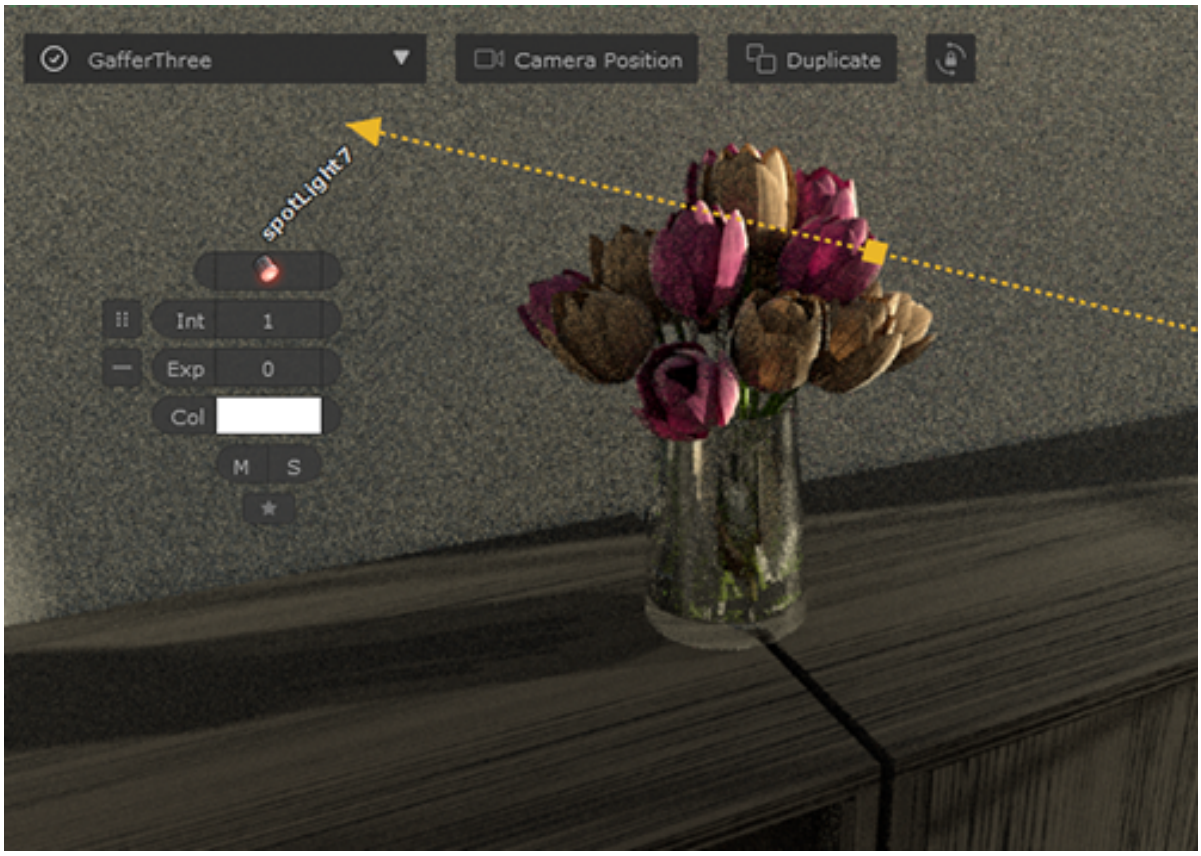
Shadow mode provides a pivot point on each light so you can control which part of geometry the light hits, and where you want the shadow cast from that geometry to fall.



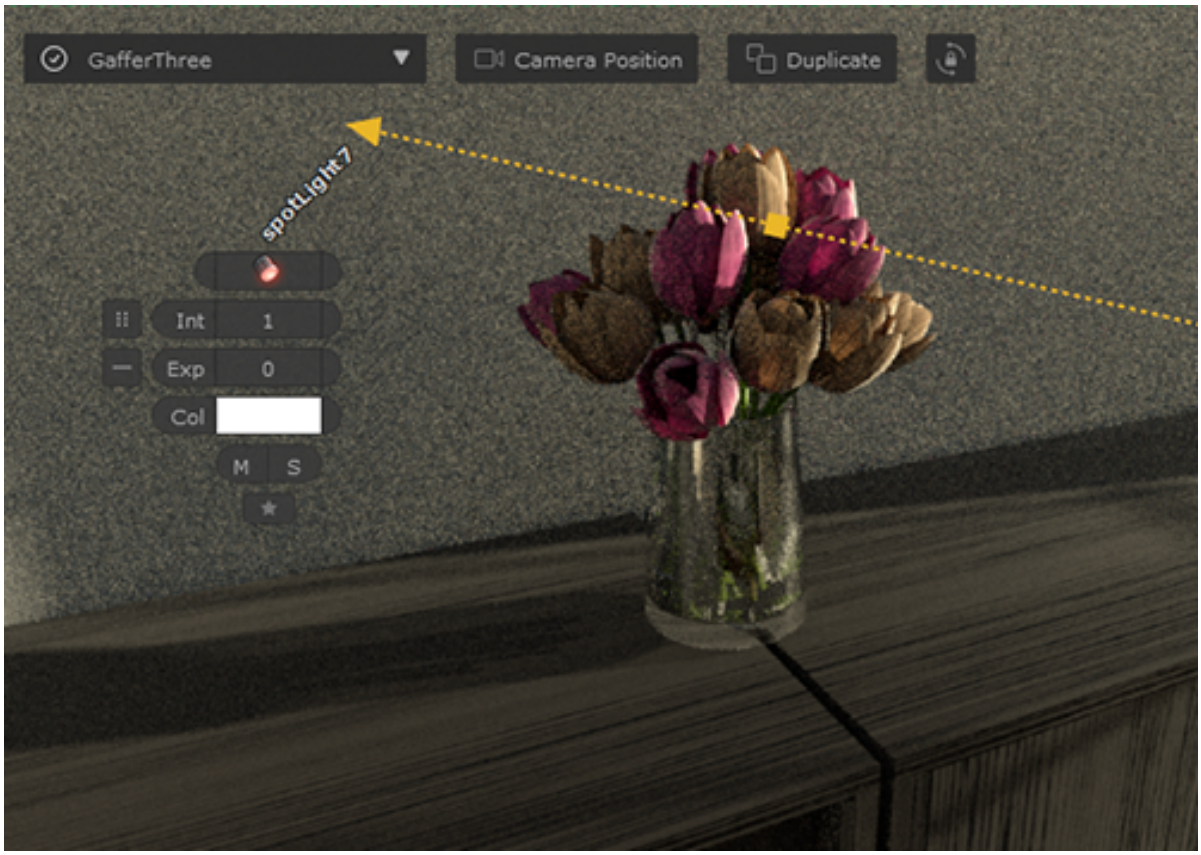
To place a light using **Shadow** Mode:

1. Select **Shadow** mode  from the **Lighting Tools** dropdown menu.
2. Choose a light type from the **Lights** dropdown menu.
3. **Shift** + **Click** to place the pivot point of the light.

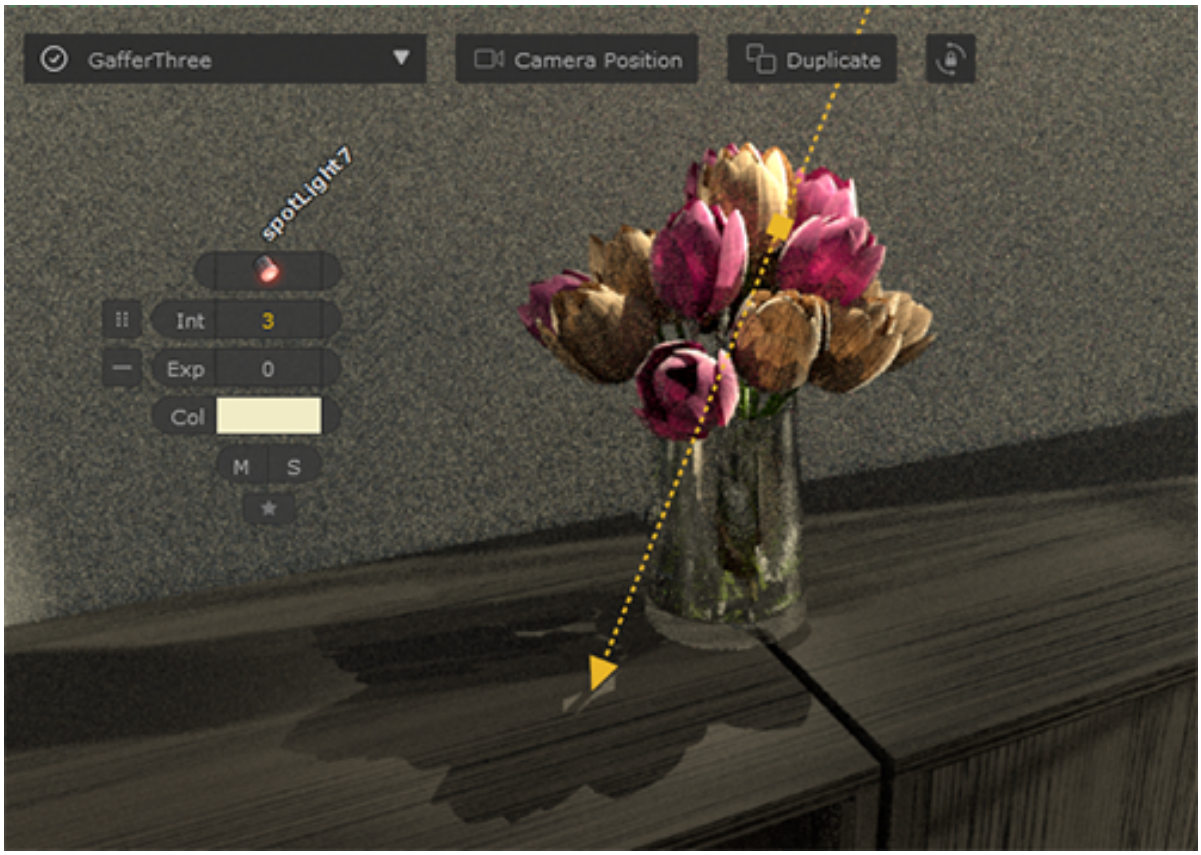
   The light is created.


GafferThree
Camera Position
Duplicate
spotLight7
Int 1
Exp 0
Col
M S

4. Click and drag the pivot point to target the section of geometry you want the light to affect.



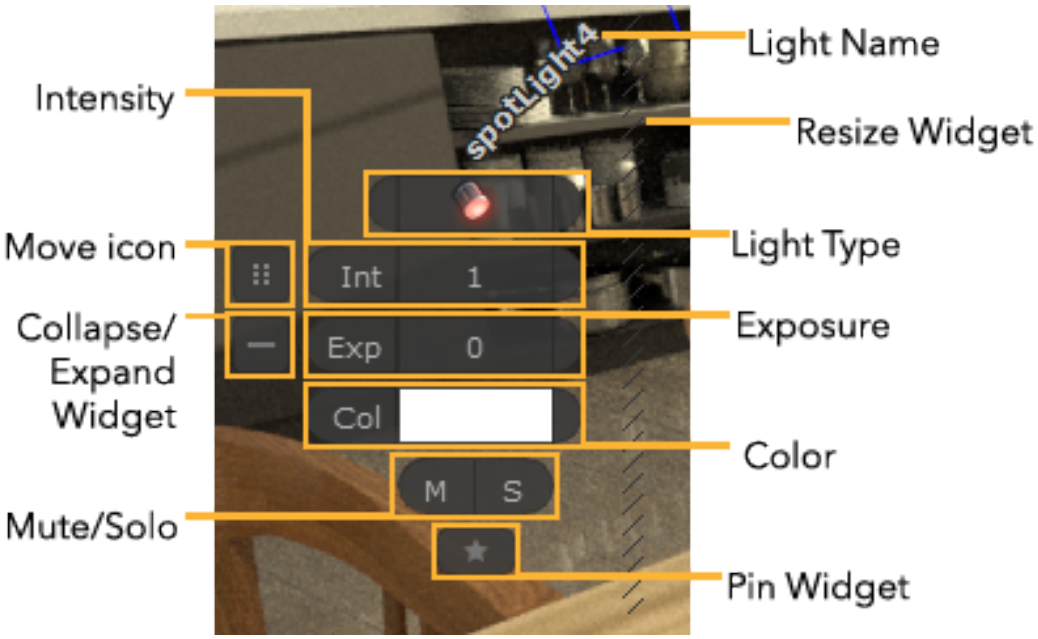
5. Click and drag the arrowhead to control where the shadow falls.



Editing Lights Using the Lighting Tools Parameter Widget

Once a light has been created, a light parameter widget is displayed containing **GafferThree** light parameters you can access and edit, without leaving the Hydra Viewer. The light parameter widget is visible while the light is selected. If multiple lights are selected, the lights are stacked in the parameter widget.

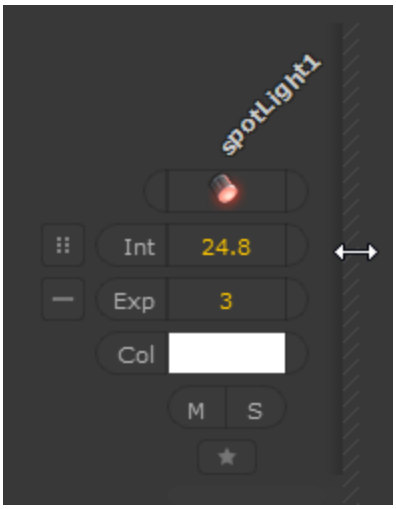
Note: To edit a light, you must have the correct GafferThree selected in the **GafferThree** dropdown, that is, where the light was created. Otherwise you can use per-shot overrides to edit a light in a different GafferThree node. For more information, see the Shot Level Overrides section of this topic.



The default light widget controls and parameters are:

- Click and drag on the move icon to move the light widget around the **Hydra Viewer**.
- Click the expand/collapse buttons to expand/collapse the light widget.
- **Int** - The light intensity.
  Click and drag in the value field to increase and decrease the intensity. You can also click and type in the value field to specify a value.
- **Exp** - The light exposure.
  Click and drag in the value field to increase and decrease the exposure. You can also click and type in the value field to specify a value.
- **Col** - The light color.
  Click on the color to open a color adjustment panel and choose a light color.
- **M** - Mute light.
  Click to toggle. When a light is muted it is omitted from renders.
- **S** - Solo light.
  Click to toggle. Only soloed lights are included in renders. All lights which are not solo-ed are muted and therefore omitted from renders.
- Click the star button to pin a light parameter widget, preventing the light parameter widget from closing when the light is deselected.

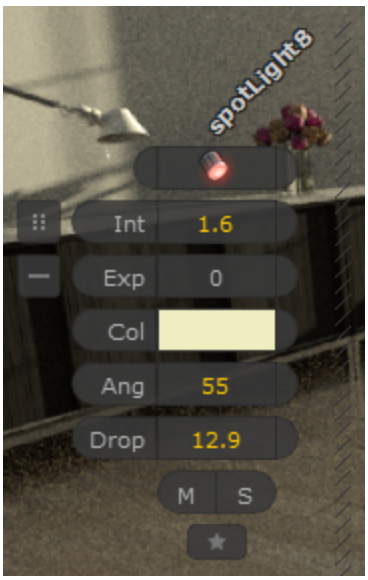
- Click and drag the hatched column on the right of the widget to resize it.



Note: For full information about the **GafferThree** light parameters, see Getting to Grips with the GafferThree Node and the GafferThree reference guide.

Customizing the Light Widget in the Lighting Tools

The light parameters widget provides quick access to the **Intensity**, **Exposure** and **Color** parameters for your lights. You may want to customize this widget to display more parameters, specific to the light type you create. This allows you to further streamline your workflow by adding the parameters you use the most.



A customized light parameter widget

The light parameters widget can be customized using the **ViewerObjectSettings** node.

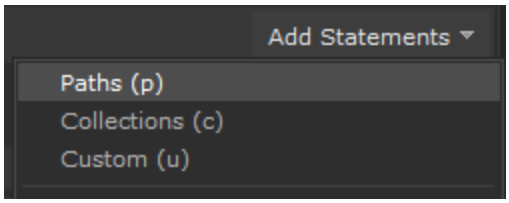


Note: You can also choose to hide certain lights' arrowheads, when they are not selected, using the **hideHandle** parameter in a **ViewerObjectSettings** node parameters.
For more information, see the ViewerObjectSettings reference guide.

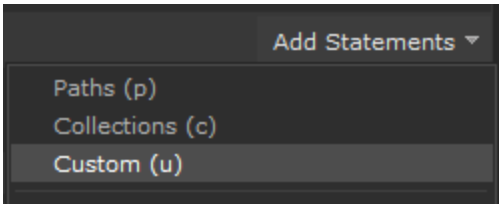
1. Create a **ViewerObjectSettings** node and place it in your **Node Graph** underneath your **GafferThree** node.



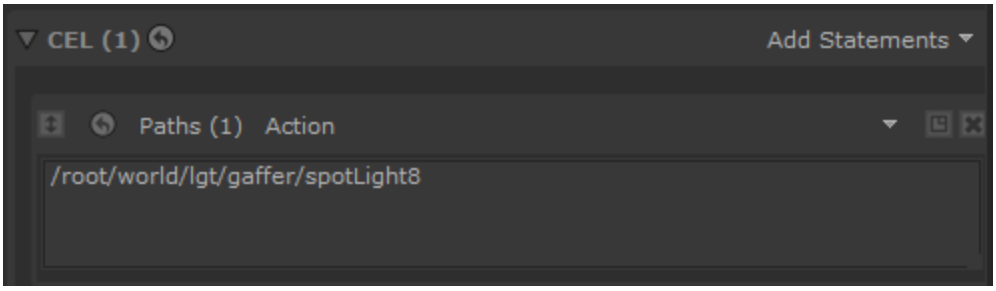
2. Open the **ViewerObjectSettings Parameters**.
3. If you want to affect just one light, click **Add Statements** > **Paths**.



If you want this statement to affect all new and existing lights of a specific type in your scene, you can use an expression to access the attribute defining the light type. For this select **Add Statements** > **Custom**.

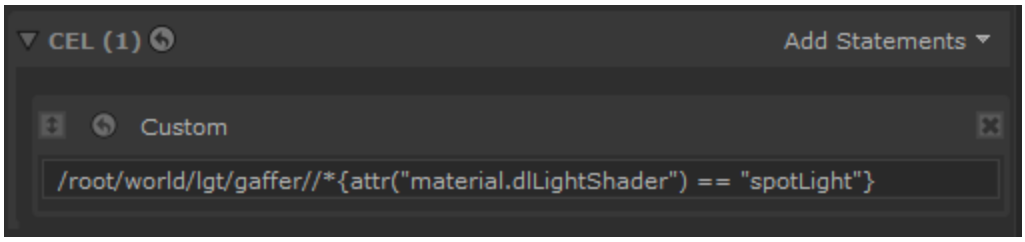


4. For single lights, **MMB** drag the Scene Graph location for the light you want to be affected, into the **Paths** field.



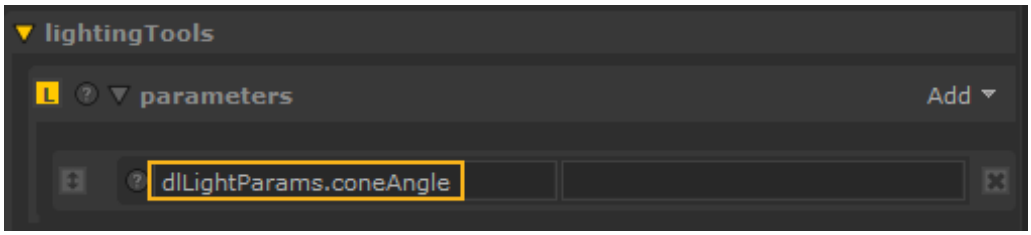
In this example, only **spotLight8** is affected.

For multiple lights, type an expression for the light types you want to be affected.



In this example, the expression targets all lights under **/root/world/lgt/gaffer/** with a **material.dlLightShader** attribute called **spotLight**. Therefore, all new and existing 3Delight **Spot Lights** are affected.

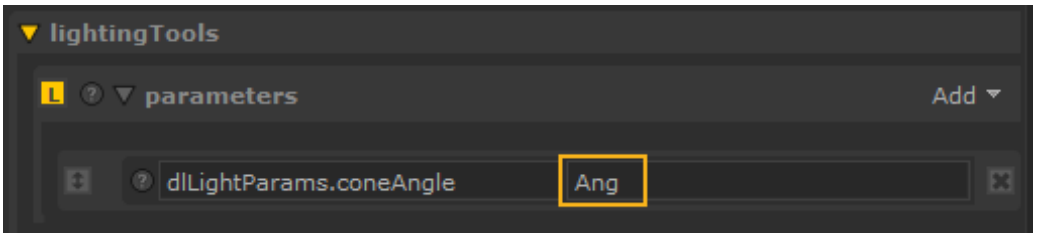
5. Navigate to the **Lighting Tools** section in the **ViewerObjectSettings Parameters**.
6. Under **parameters**, in the first text field, enter the path of the parameter you would like to add to the light parameter widget.



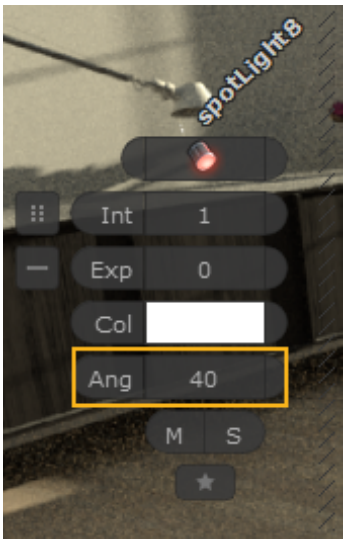
The parameter is now accessible from the parameter widgets.



7. In the second text field, enter the label you would like to appear on the light parameter widget for this parameter.



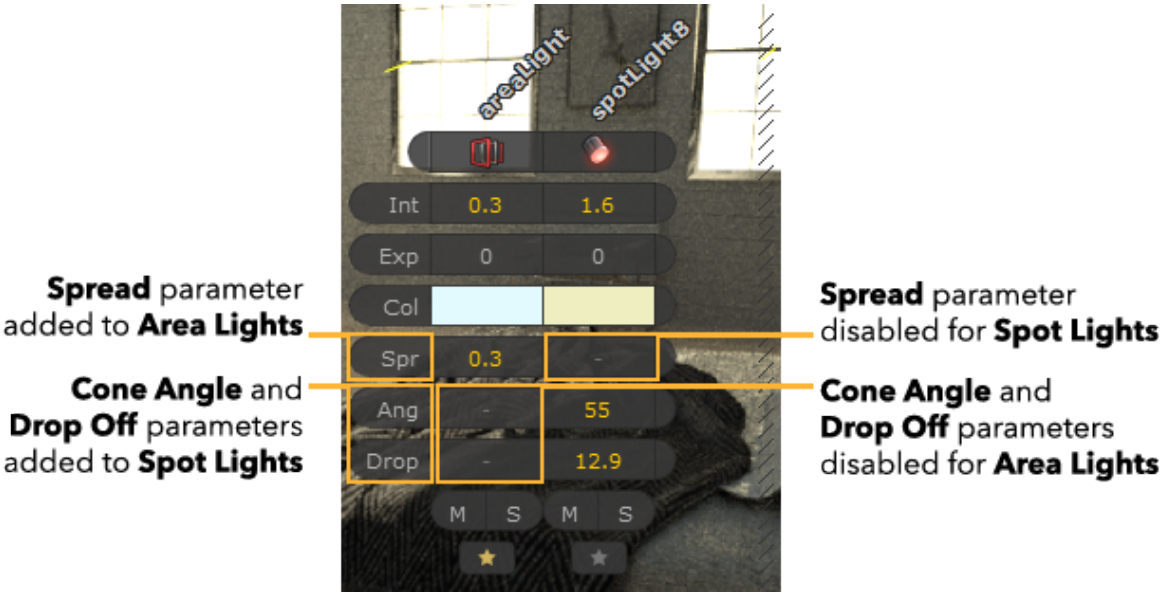
The label is updated on the light parameter widget.



> **Note:** The label field is optional. If a label is not provided, the parameter's original label is used.

You can add as many extra parameters as needed to allow you to work efficiently when using Lighting Tools.

If you have two lights selected and have added extra parameters, the parameters for all the lights are visible but you can only edit the parameters for the light types that they exist in. If a parameter does not exist in a certain light type, it is disabled for that light type and marked with a dash.



Shot Level Overrides

It is common to work with multiple GafferThree nodes in one scene. This enables artists to set up lighting for the entire sequence in one GafferThree node, and make per-shot overrides in other GafferThree nodes.

Lighting Tools makes it easy to switch between different GafferThree nodes, they are all displayed in a dropdown list in the **Hydra Viewer** when Lighting Tools is enabled.

Depending on which light is selected, a symbol is displayed next to the selected GafferThree.

	The selected lights can be edited and all edits can be recorded in the selected GafferThree node.
	The selected lights parameters cannot be edited in the selected GafferThree node as it does not have the ability to record edits. The selected lights can be edited in an

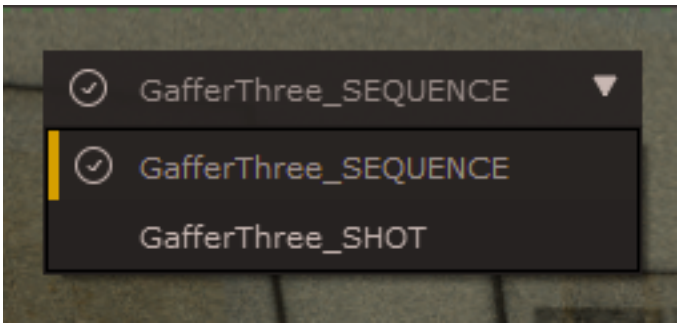
	upstream GafferThree node. Transform edits can be made from the selected GafferThree node and recorded in the upstream GafferThree node. Parameter edits, such as **Intensity** and **Exposure**, can only be made in an upstream GafferThree node.
	The selected lights cannot be edited either in this GafferThree node or any upstream GafferThree node. This means the selected light has no creation or edit package, or it is inside a locked node, such as a LiveGroup.
	This symbol represents a combination of the  and  symbols. This means one or more selected lights can be edited in the selected GafferThree node, and one or more selected lights can be edited in an upstream GafferThree node.
	This symbol represents a combination of the  and  symbols. This means one or more selected lights can be edited in the selected GafferThree node, and one or more selected lights cannot be edited in this, or any upstream GafferThree node.
	This symbol represents a combination of the  and  symbols. This means one or more selected lights can be edited in an upstream GafferThree node, and one or more selected lights cannot be edited in this, or any upstream GafferThree node.
	This symbol represents a combination of the  ,  and  symbols. This means one or more selected lights can be edited in the selected GafferThree node, one or more selected lights can be edited in an upstream GafferThree node, and one or more selected lights cannot be edited in this, or any upstream GafferThree node. This symbol implies there are more than three lights selected.

To edit a light outside of it's original **GafferThree** node:

1. Select the light you want to edit.



2. In the **GafferThree** dropdown, choose the **GafferThree** you want your edits to be recorded in.

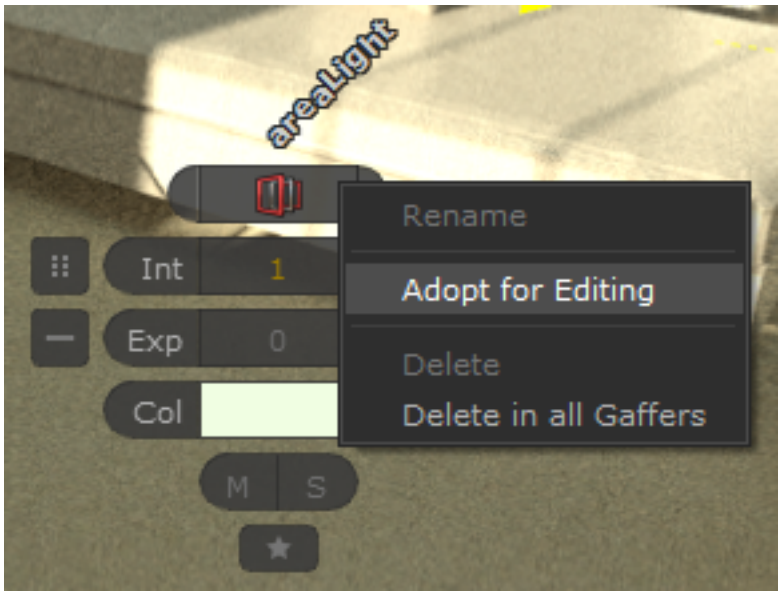


Note: The **GafferThree** you choose must be downstream from the original **GafferThree** node.

The GafferThree node is marked with a symbol.



3. **Right**-**Click** the light parameter widget and select **Adopt for Editing**.

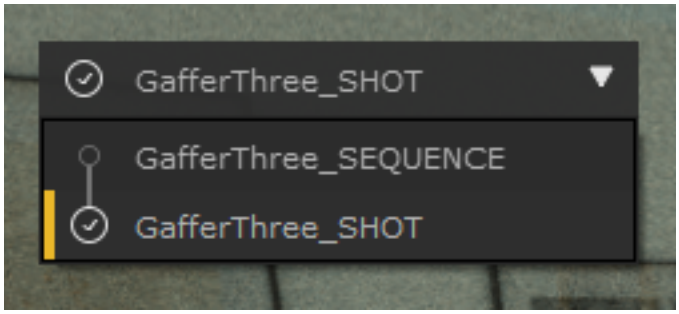


The GafferThree node is marked with a symbol and you can now make edits to the light through the parameter widget.



Note: These edits do not affect the light parameters in the original **GafferThree** node.

The **GafferThree** dropdown provides a breadcrumb trail of edits so it is clear what changes have been made.



Cloning Lights and Using Template Materials With Lighting Tools

When lighting a scene, artists often want to copy existing lights to place multiple versions with the same properties in various locations. For example, a room may have ten lights on the ceiling, each with the same properties.

Depending on the situation, an artist may want to simply use an existing light as a basis when creating a new light, or use a **Template Material**. **Template Materials** allow you to define the overall look of the lighting in a scene, while making local overrides per light if necessary. Assigning a **Template Material** to multiple lights means that you only need to change the parameters of the **Template Material**, rather than each individual light.



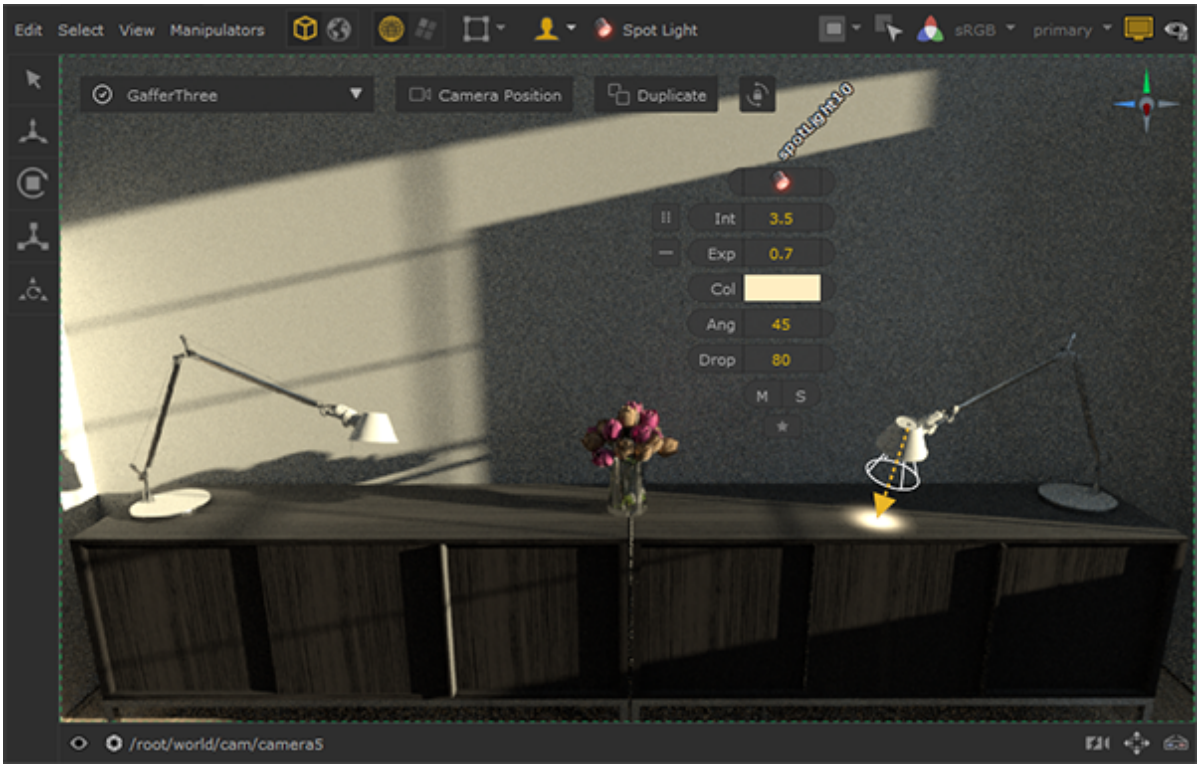
Note: For more information on Template Materials, see Defining a Template Light Material.

Katana's Lighting Tools allows you to clone lights using the **Clone** and **Clone as Template** light options. These options create new lights with the same parameters as the light you cloned.

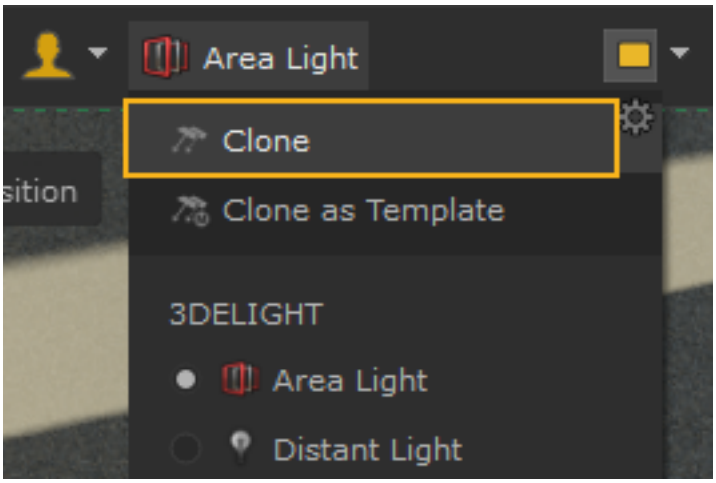
Cloning a Light

The **Clone** option uses a selected light as a basis for the new light you create. The new light has all the same parameters as the selected light including its size, distance, intensity, exposure and color, and any other parameters you may have changed.

In this example we have two lamp objects and want to place **Spot Lights** to resemble the light bulbs. One **Spot Light** is set up so we can use the **Clone** option to create the second light.



1. Enable **Lighting Tools**  in the **Viewer**.
2. Choose the **Clone** option from the Lights drop-down menu.



3. Select the light you want to clone.

4. **Shift** + **Click** to place a light under the lamp.



A new **Spot Light** is created with the same properties and parameters as the cloned **Spot Light**.

5. Use the manipulators to position the new **Spot Light**.



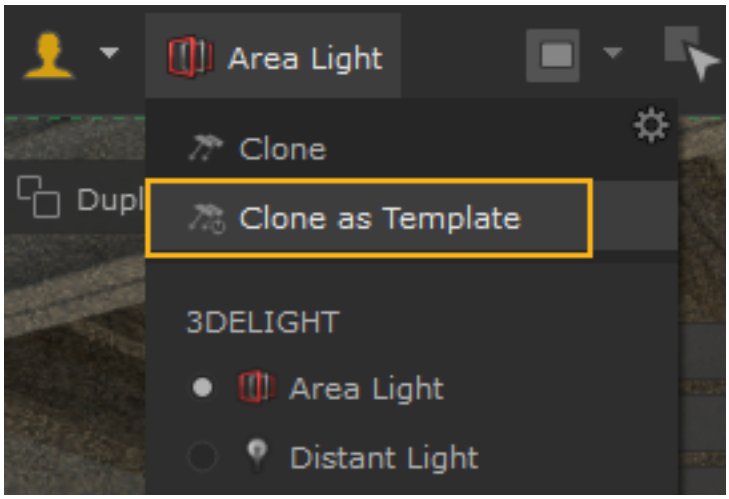
How to Clone a Light as a Template

The **Clone as Template** option lets you use a selected light as a basis for a **Template Material**. The new light, and selected light are both assigned to that **Template Material**.

In this example we have a series of windows and we want to place an **Area Light** behind each one to resemble sunlight. The first window already has an **Area Light** so we can use the **Clone as Template** option to create a **Template Material** from the existing light, and place new lights which are also assigned to the **Template Material**.



1. Enable **Lighting Tools** in the **Viewer**.
2. Choose the **Clone as Template** option from the **Lights** drop-down menu.



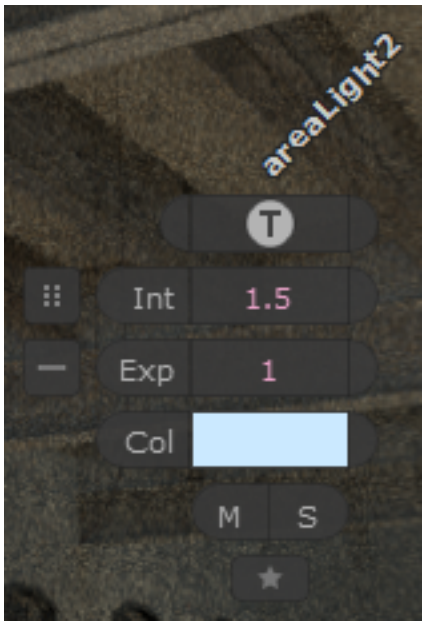
3. Select the light you want to clone as a **Template Material**.
4. **Shift** + **Click** to place a new light.

   A **Template Material** is created using the parameters of the selected light.

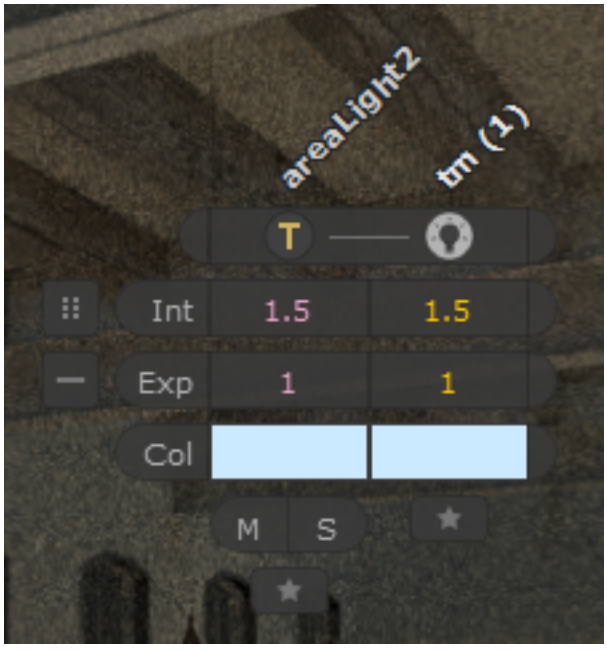
   A new light is created and assigned to the new **Template Material**.

   The original light is assigned to the new **Template Material**.

   Any light that is assigned to a **Template Material** is displayed with a **Template Material** button on the parameter widget, and any parameters driven by the **Template Material** are pink.



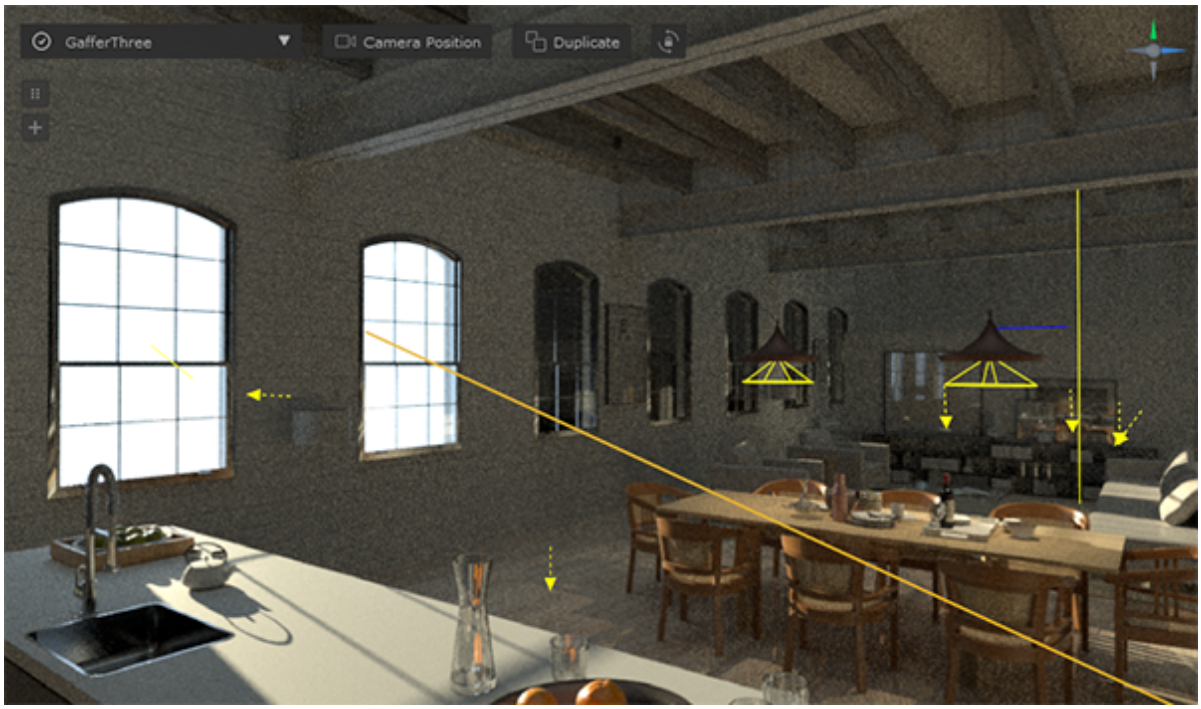
Click the **Template Material** button  to show the parameters for the **Template Material**.



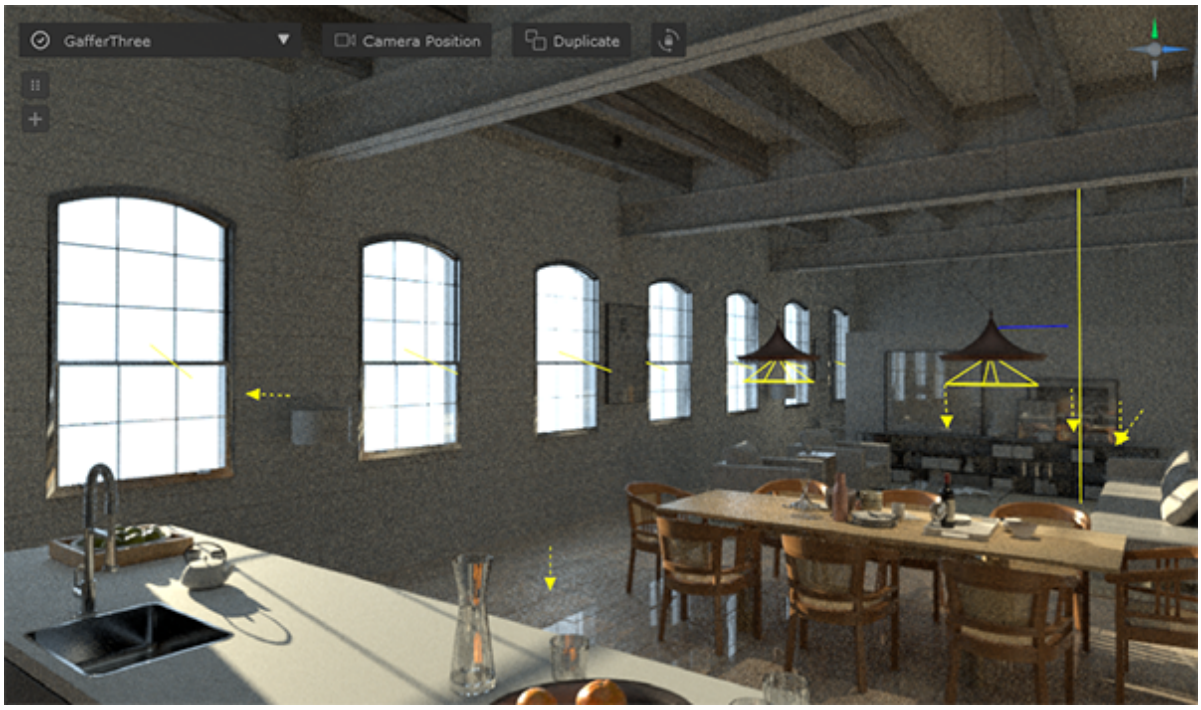
Click the  button on the Template Material parameters to show all other lights assigned to that **Template Material**.



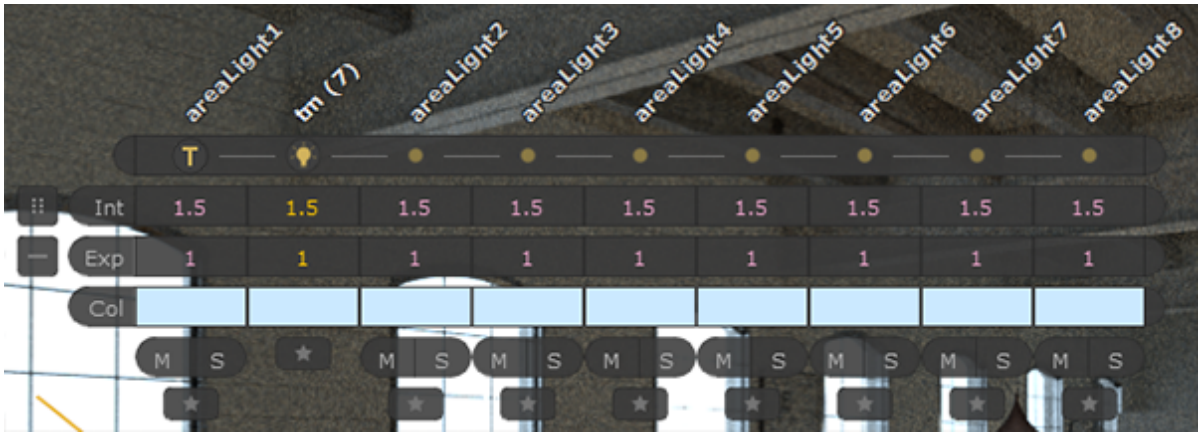
5. Position the new light behind the next window.



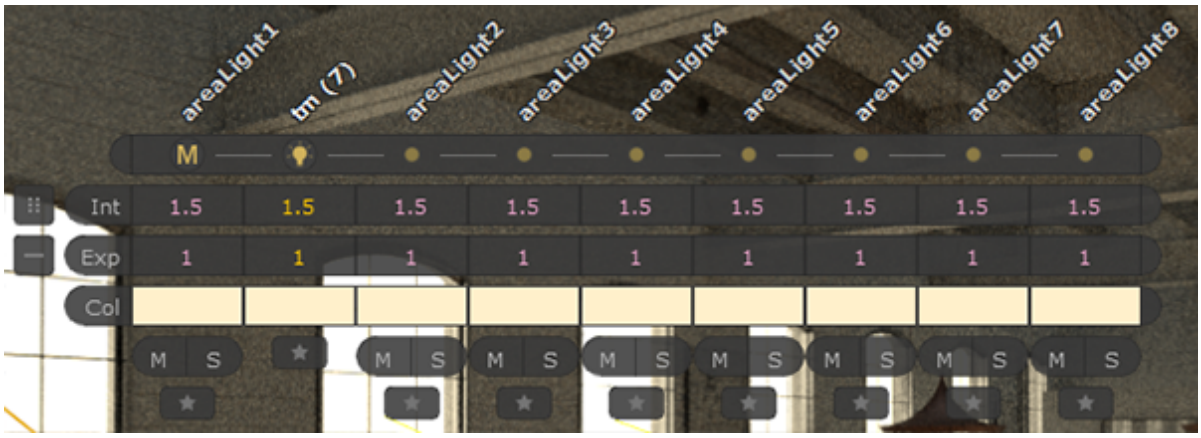
6. You can now **Clone** or **Duplicate** these lights to place behind the remaining windows.



The new lights are also assigned to the **Template Material**.



7. Once all the lights are set up, any changes you make to the **Template Material** are also applied to the lights.


areaLight1
tm (7)
areaLight2
areaLight3
areaLight4
areaLight5
areaLight6
areaLight7
areaLight8
M
Int
1.5
Exp
1
Col
M
S



Look Development

This chapter walks you through how to use Katana as a Look Development tool.

Look Development with Look Files - An introduction to look files and how they are used to store the changes from one state of the scene graph to another.

Adding and Assigning Materials - A material is a scene graph location that holds shaders.

Checking UVs - Use the UV Viewer tab to inspect the UVs of selected objects.

Look Files - More uses of Look Files in Katana including baking, material overrides, collections, and palettes of materials.

Look Development with Look Files

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 Node Parameter Basics.

6. Enter the asset name for the look file in the **saveTo** parameter.
7. Click the **Write Look File** button.

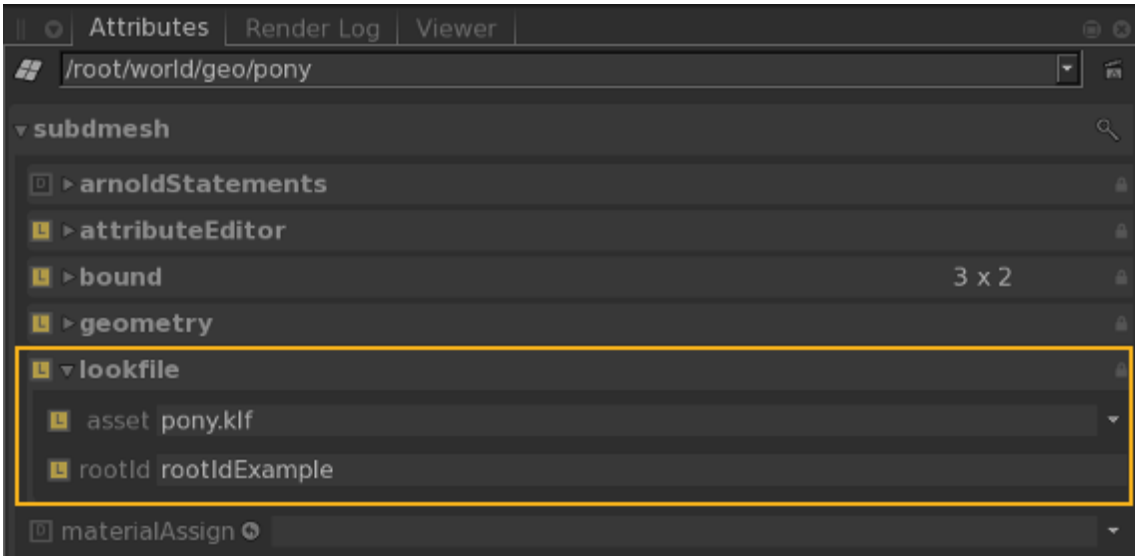
   The **Write Look File** dialog displays.
8. Select where the asset is going to be saved (it defaults to the **saveTo** parameter) and click **Accept**.

   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.

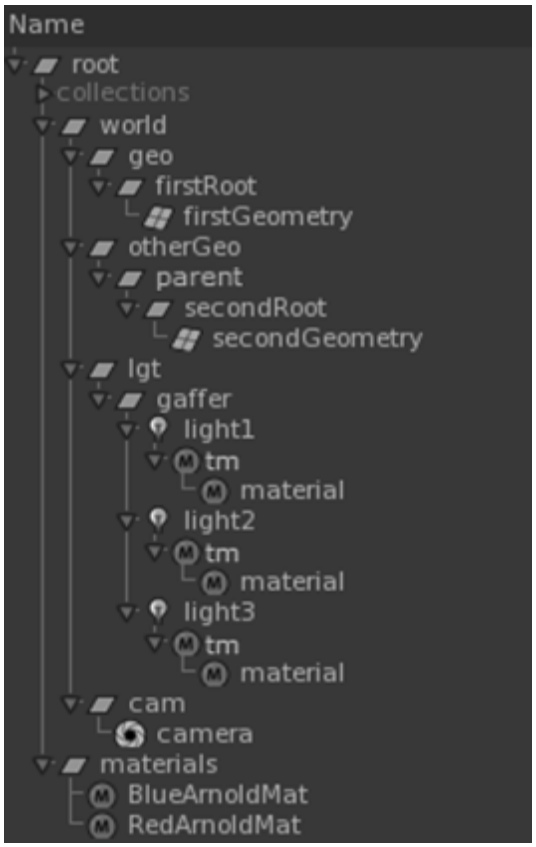
Note: You can select any scene graph location as a root location. It determines the first level of the relative path - or paths - generated by a LookFileBake.

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.

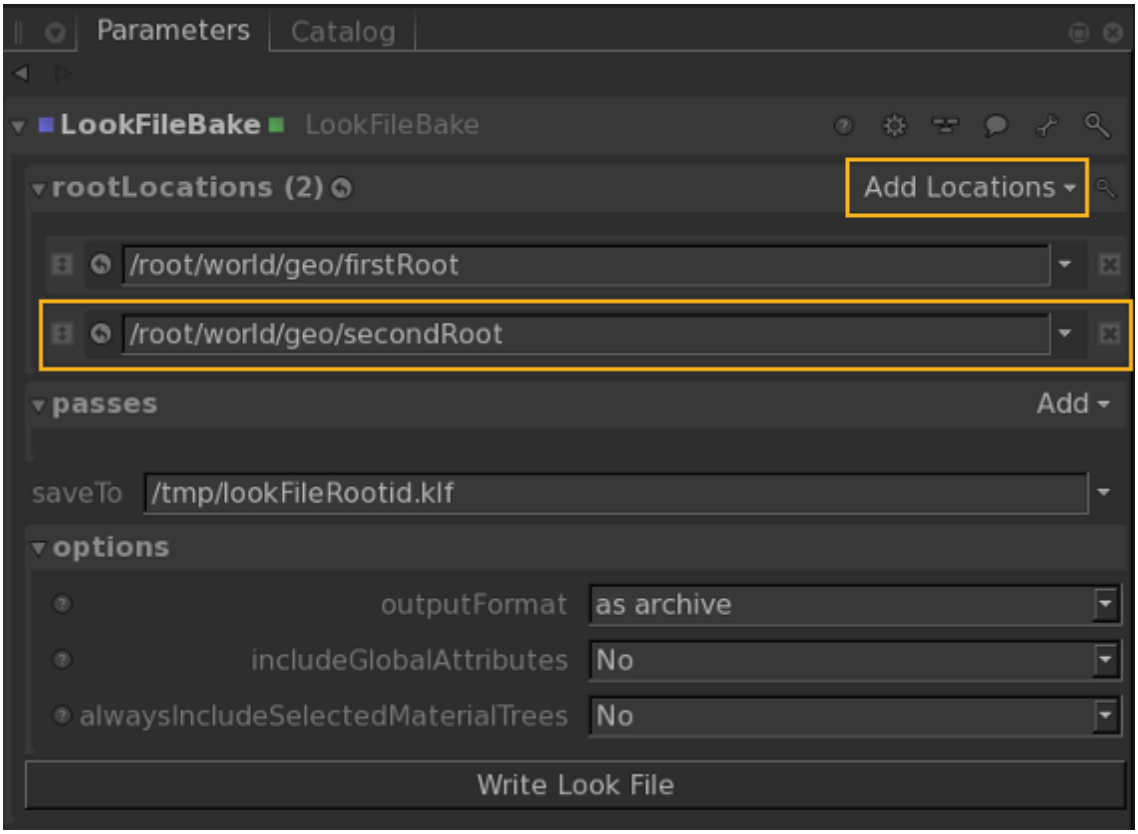
The scene graph shown below has multiple root locations **firstRoot** and **secondRoot**, with geometry under each.



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 Development with Look Files 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.

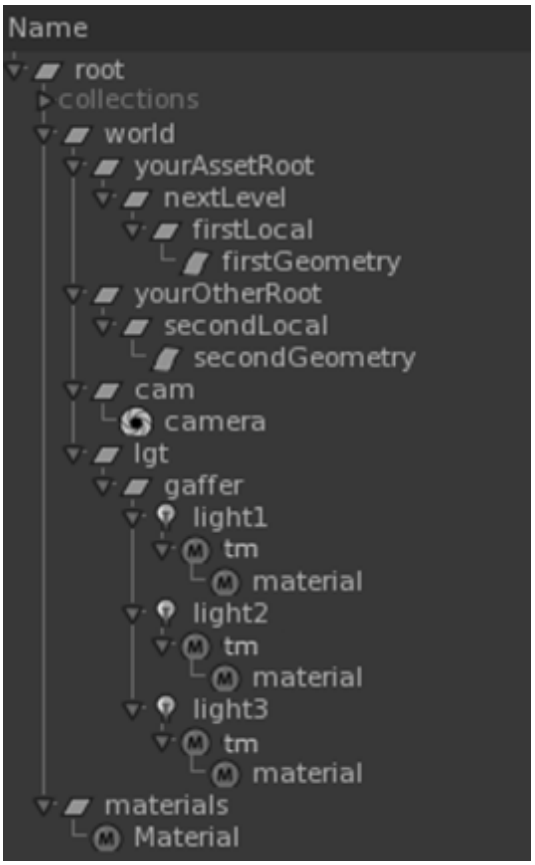


• Write the Look File.

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 Node Parameter Basics).
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 it 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.

   If no look file pass is specified when attempting to resolve, the pass falls back to the default pass when **KATANA_LOOKFILE_DEFAULT_PASS_FALLBACK** is set to **1**. In this case, no error is generated and the default pass is used, otherwise Katana produces an error location.



Note: The **lookfile.resolvedPass** attribute always reports the requested pass and is, therefore, not affected by this fall-back.



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).

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 was as the LookFileResolve node. The LookFileManager node can resolve multiple passes, providing an output for each.

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.
- Adding a look file from all the look files in the current scene graph.
- Adding a look file from a list of all look files at or below a scene graph location.
- Adding a look file that is not assigned anywhere within the scene graph.

To add the look file currently assigned to a scene graph location, right-click inside the **Look Files** list (or click ) and select **Add Look File Asset From Scene graph Selection**.

To add 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.

To add 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.

To add 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

You can replicate the behavior of the LookFileGlobalsAssign node inside the LookFileManager.

Assigning a Global Look File

To assign 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**.

To assign 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.

To override or edit 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.

You can toggle the ignore state of an override by right-clicking on the override in the **Add Override** list (or selecting it and clicking) and selecting **Toggle Ignore State**.

Duplicating an Existing Override

To duplicate an 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

To view override parameters in another panel, right-click on the override in the **Add Override** list (or select it and click), and select **Tearoff Parameters of Override...** .

Deleting an Override

To delete an override, right-click on the override in the **Add Override** list (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.

Adding and Assigning Materials

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.

Material Basics



Note: The following steps are part of a legacy workflow, as of Katana3.2 you can create materials using NetworkMaterialCreate nodes. For more information and workflow examples, see Building Materials Using NetworkMaterialCreate.

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 Adding Multiple Materials for more information.

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 Material Basics 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.
5. 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.

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 Parameterfor 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 overriden 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).

Assigning Materials and Textures

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.

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 more information on textures, refer to Handling Textures.

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, seeTurning 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**.**

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.

Material Pipelines

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:

1. Follow steps 1 to 4 in Material Basics above to create a material location.
2. Select **create from Look File** in the **action** parameter dropdown.
3. Enter the path to the look file in the **lookfile** parameter, or click ▼ > **Browse...**, navigate to the look file and click **Accept**.
4. 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.
5. 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.
6. 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:

1. Follow steps 1 to 4 in Material Basics above to create a material location.
2. 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.

Adding Multiple Materials

Having a chain of Material nodes would soon clutter up a recipe. To avoid this, create multiple materials within one node using 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.

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 .

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.

To add Material nodes from the **Node Graph** into the MaterialStack node, **Shift**+middle-click and drag the nodes into the **Add** list.

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**).

Moving Materials Within the Add List

To move materials within the **Add** list, middle-click and drag.

NMX Utility Nodes

These nodes and operations have been added as an extension to using the NetworkMaterialCreate, NetworkMaterialEdit, NetworkMaterialSplice and Material nodes in combination with each other.

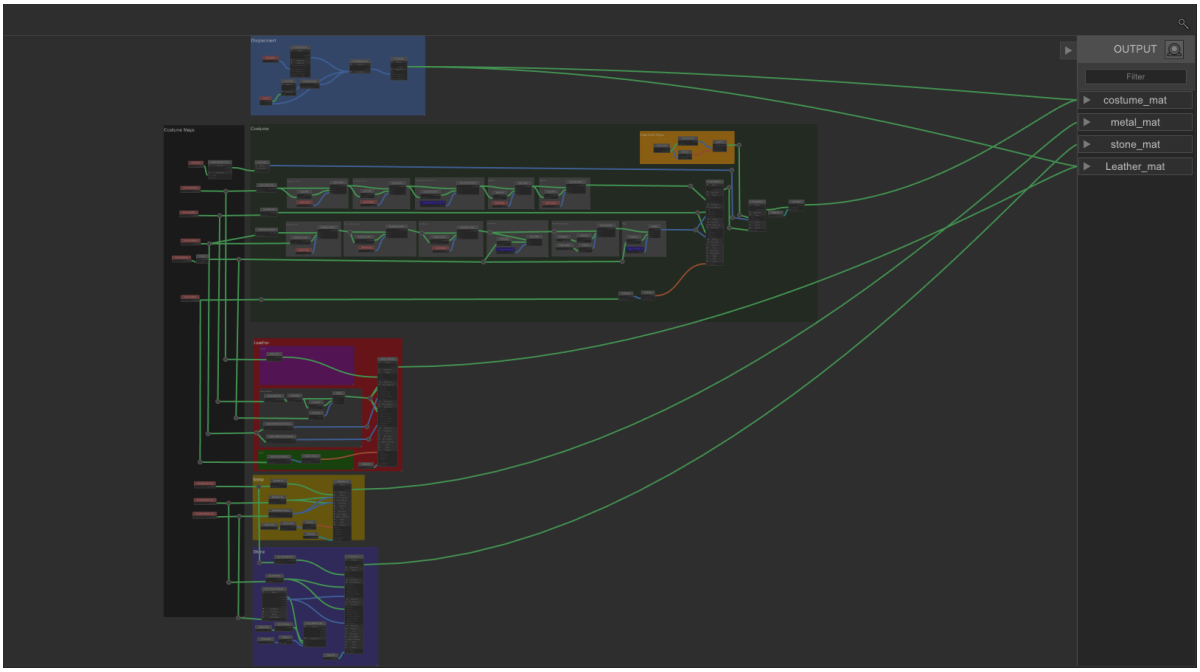
Removing non-contributing nodes from the network

The **NetworkMaterialLayoutFilter** can be used to remove any nodes that aren't contributing to a material (that have been created and edited within the NetworkMaterialCreate and NetworkMaterialEdit nodes). This can improve the performance of the NetworkMaterialEdit node when editing an incoming material, since it needs to process and draw fewer shaders.

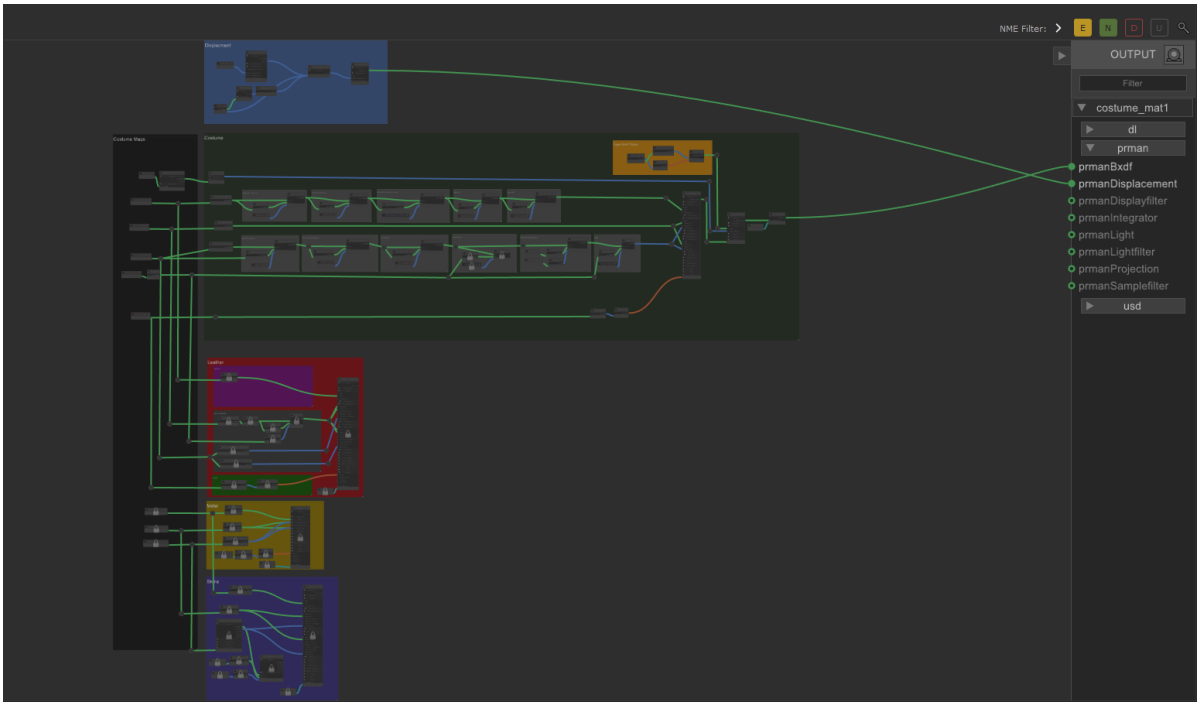
This is particularly helpful when editing a material that had been originally created in a multi-material NetworkMaterialCreate node, which may contain many nodes that are not relevant to the edited material. In this case, when these materials are edited in a downstream NetworkMaterialEdit, other shaders pertaining to other materials will also show up and appear as locked nodes.

In the example below, multiple materials have been created in a NetworkMaterialCreate. When the "costume_mat" material is edited in a downstream NetworkMaterialEdit, the nodes that aren't affecting "costume_mat" will still be shown, in a locked state.

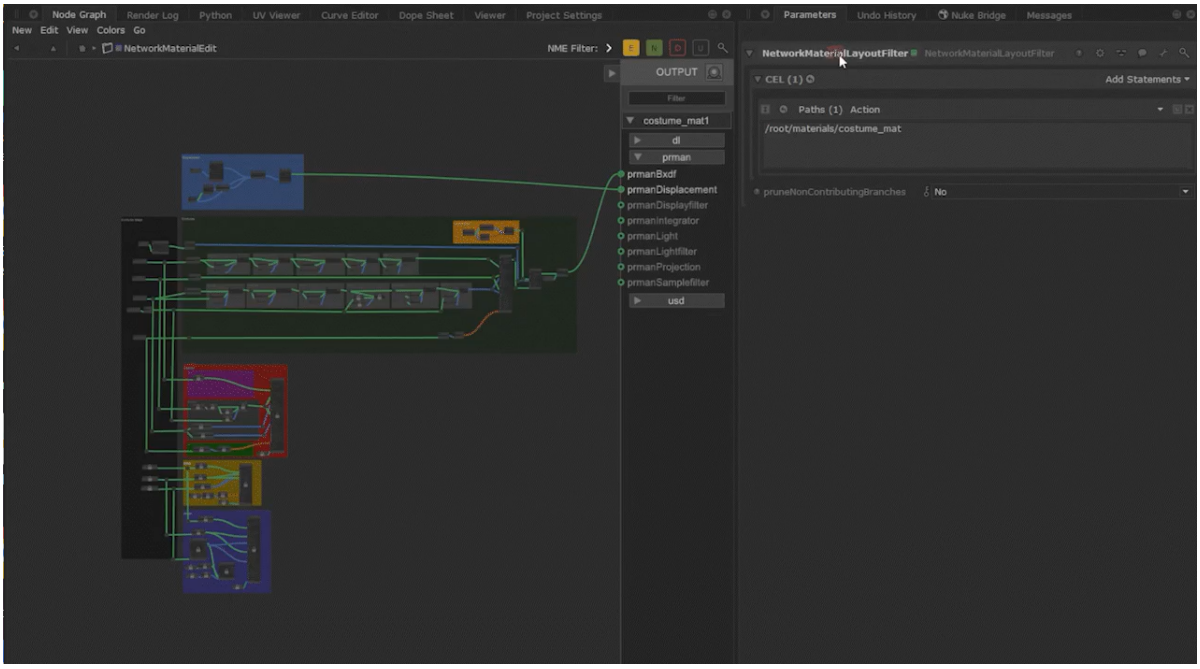
NetworkMaterialCreate node



NetworkMaterialEdit node



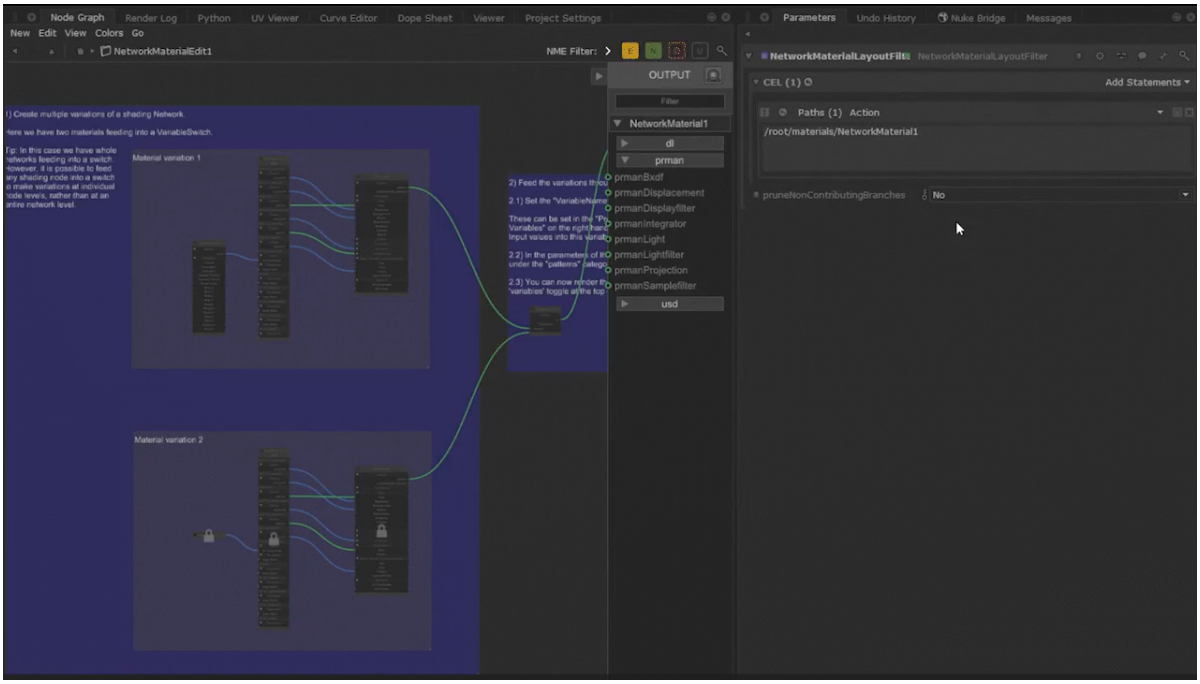
With the use of the **NetworkMaterialLayoutFilter** node, these unconnected nodes will no longer appear. See below:



Backdrop nodes, since they're not elements that are connected to the material, are preserved unless they are within a group that is filtered out.

By default, these shading nodes will be pruned based off of their connection to the material's output. However, when **pruneNonContributingBranches** is set to **Yes** in the node's Parameters, nodes can also be pruned based on their contribution to the final material.

An example of this is when nodes are connected to either side of a VariableSwitch node and picked up in a downstream NetworkMaterialEdit. In this use case, the branch that isn't currently being used will contain locked nodes. See example below:



Applying Material node attributes to the NetworkMaterialEdit context

Both the Material node and NetworkMaterialEdit node pick up existing Material Interface information and allow you to further modify them. However, when changes are made in a Material node, they're not reflected in a downstream NetworkMaterialEdit node by default.

To improve the workflow between these nodes, the **MaterialInterfaceResolve** node enables any changes in a **Material** node to be reflected in a downstream **NetworkMaterialEdit** node:

1. Add a **MaterialInterfaceResolve** node and connect it in the Node Graph downstream of a **Material** node that has seen changes.
2. Specify the path of the material to target this node to.
3. After refreshing, view that any changes made in the **Material** node become visible in the Material Interface parameters of the downstream **NetworkMaterialEdit** node.

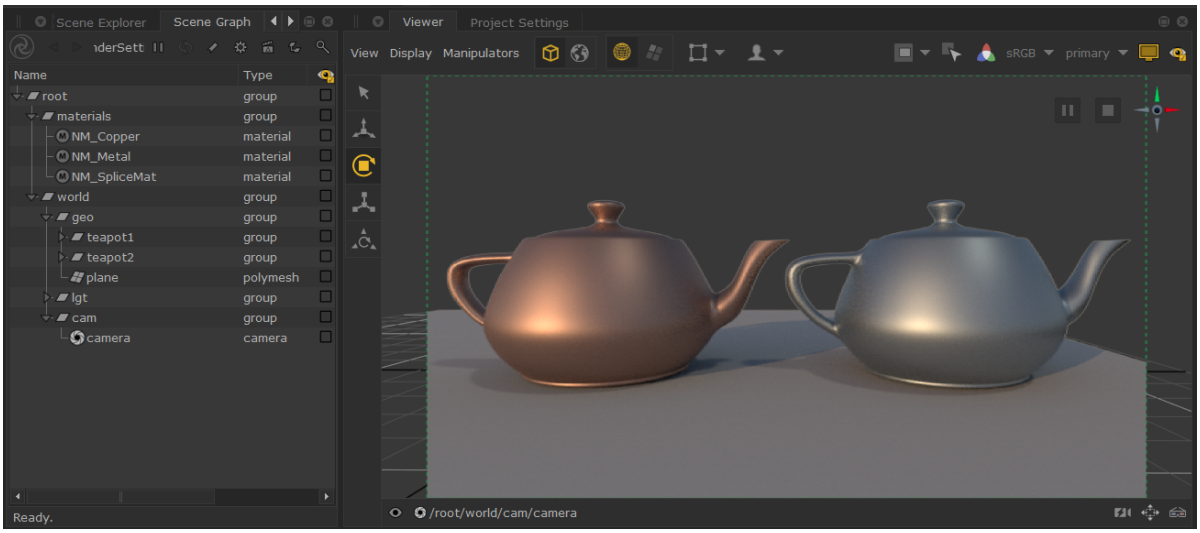
This ensures further changes to that parameter made on the **NetworkMaterialEdit** node, will be reflected on the final render.

Using advanced splicing options

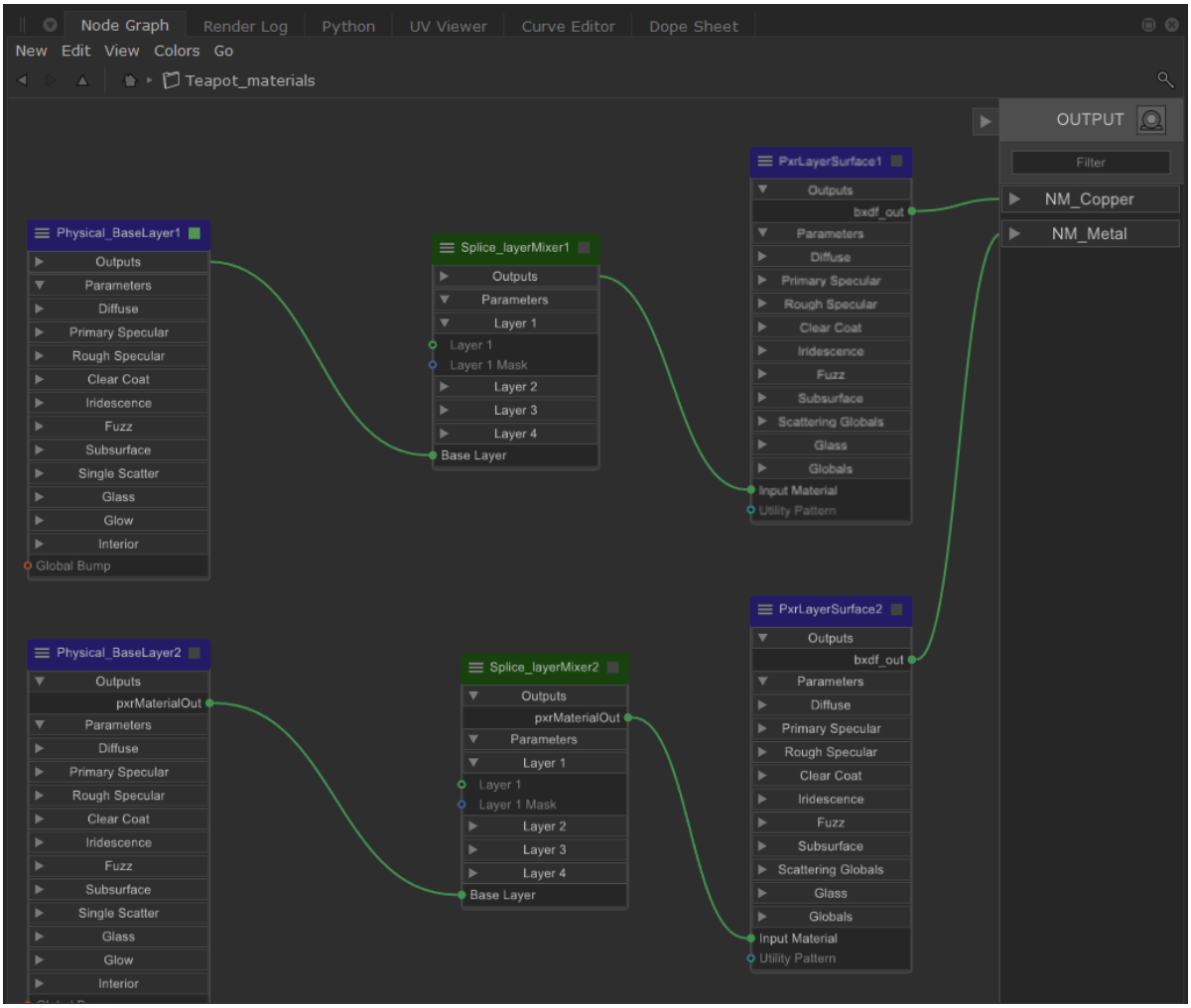
The NetworkMaterialMultiSplice node is an extension from the NetworkMaterialSplice node. The NetworkMaterialMultiSplice provides additional fields for reference material, allows CEL expressions to target multiple materials, and gives you the ability to connect wildcarded node names. This spliced material can then further be edited in a downstream NetworkMaterialEdit node. Viewing the material downstream will include the spliced nodes.

- The **materialsToEdit** > **referenceMaterial** parameter will indicate the material to appear in the node pop-up window, and the **materialsToEdit** > **spliceMaterial** parameter will allow for further materials to receive the spliced information.
- **errorOnFail**: By toggling this option, if you try to make a connection to a non-existent node, Katana will show the location error as a warning message.
- **warnOnMultipleMatches**: By toggling this option, Katana will display an error message if there are similarly named nodes in the target location.
- In **connections**, you can create multiple connections based on wildcarded names using asterisk symbol (*).
- In **disconnections**, you can disconnect wildcarded nodes in the reference material.

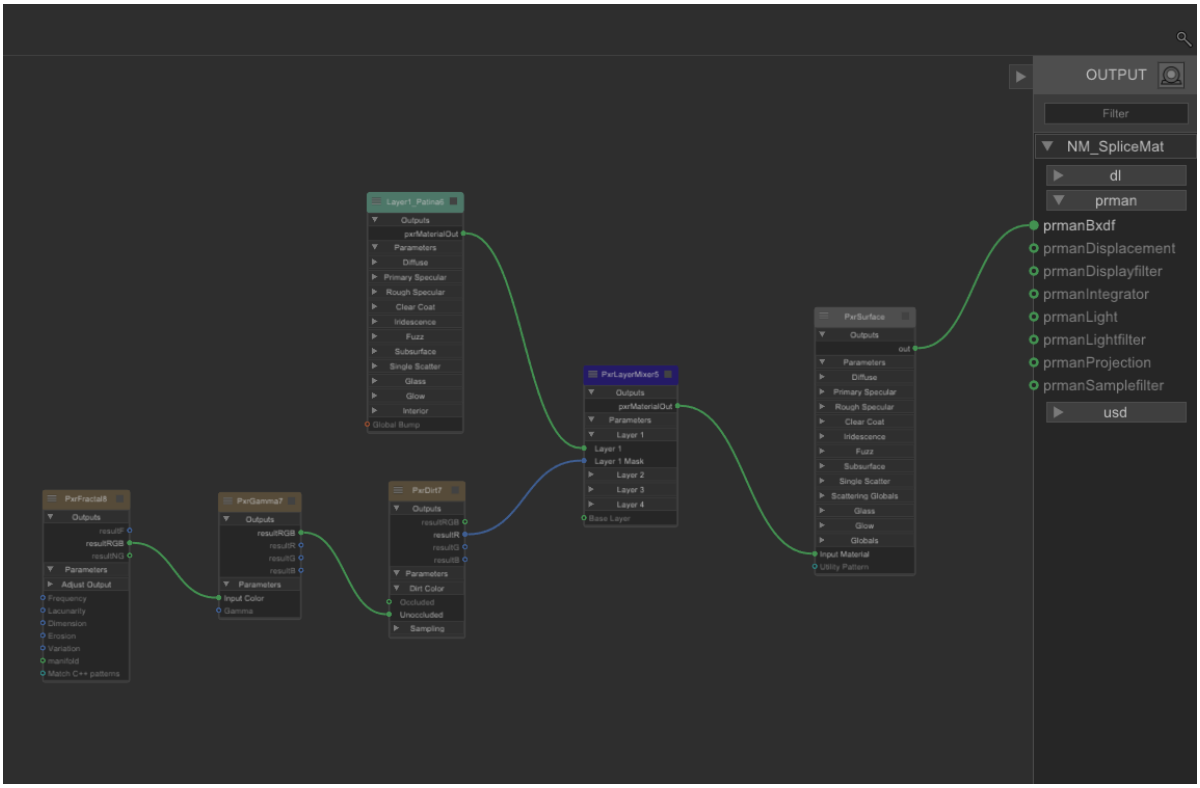
For example, some unspliced materials:



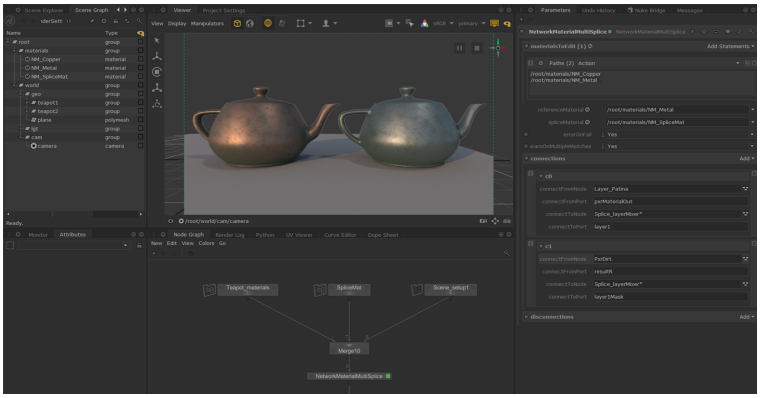
The network for two basic teapot materials:



Material to be connected/spliced:



Spliced materials (click to zoom into the image):



Tip: You can view a preview of the new material network by clicking in the Parameters tab.



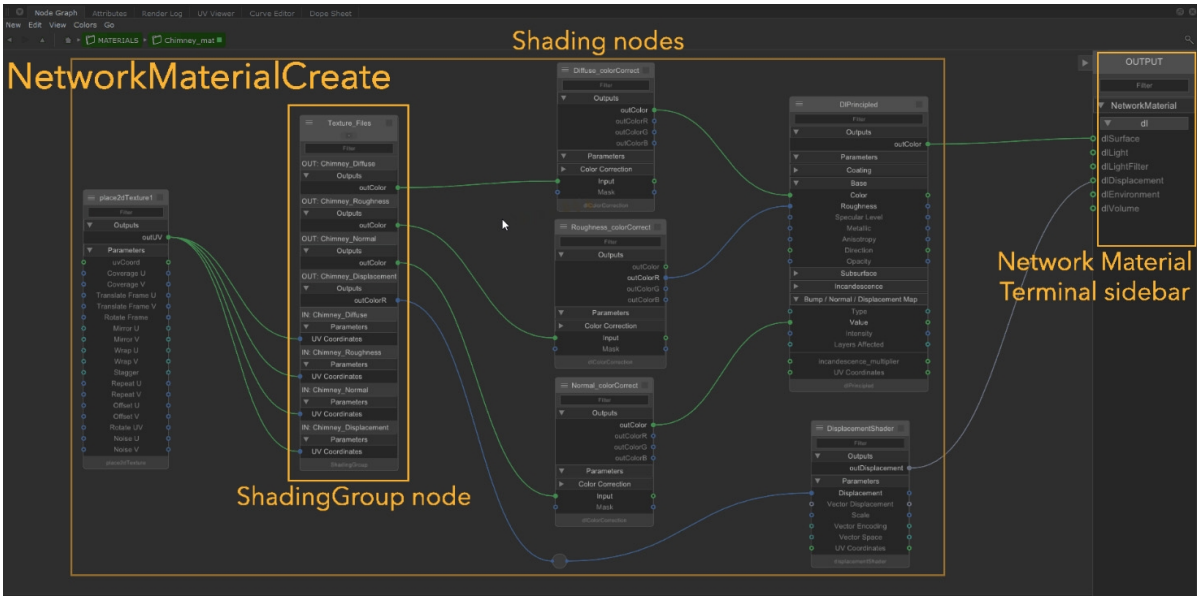
Building Materials Using NetworkMaterialCreate

The NetworkMaterialCreate node is specifically designed for building materials. It is similar to a Group node as it acts like a container for a selection of your node graph, however it exclusively stores your material network.

Note: To learn about the NetworkMaterialCreate node parameters, see NetworkMaterialCreate.

One aim of the NetworkMaterialCreate node is to minimize the amount of separate nodes the user needs to create when working with materials. Because of this, it incorporates a NetworkMaterial and other nodes, all conveniently within the NetworkMaterialCreate node. The node features a left-to-right workflow and a new shading node design, which enables you to work more efficiently, making building and editing materials as quick and simple as possible. NetworkMaterialCreate nodes support multiple NetworkMaterial locations to further streamline your workflow.

This new network material workflow also introduces the ShadingGroup node which allows you to section off pieces of your shading network within a NetworkMaterialCreate node. This layout results in multiple levels of networks which allows full control over the accessibility of certain shading nodes and parameters.



Example node graph showing the NetworkMaterialCreate workflow layout

Creating Shading Networks

Learn how to use the NetworkMaterialCreate node to build shading node networks.

Adding Multiple NetworkMaterials

Use the NetworkMaterialCreate node to create and organise multiple NetworkMaterials.

Using the ShadingGroup Node

Learn to use the ShadingGroup node to keep your shading node networks organized.

The Node Parameters and Interface Controls

Use the Node Parameters and Interface controls to customize shading node parameters from outside the NetworkMaterialCreate node.

The NetworkMaterialEdit Node

Edit NetworkMaterials that have been created using NetworkMaterialEdit nodes.

NetworkMaterialCreate Compatibility

The new workflow of using a NetworkMaterialCreate node to build a material is both forward and backward compatible.

This means that your shading node networks from previous Katana versions can be copy-and-pasted into a NetworkMaterialCreate node and the shading nodes will appear in the updated node design. The network will be connected up correctly and will give you the same result as before.

In the same way, the shading nodes from your new network within the NetworkMaterialCreate can be copy-and-pasted to previous versions of Katana as well as copied to the root of your node graph.

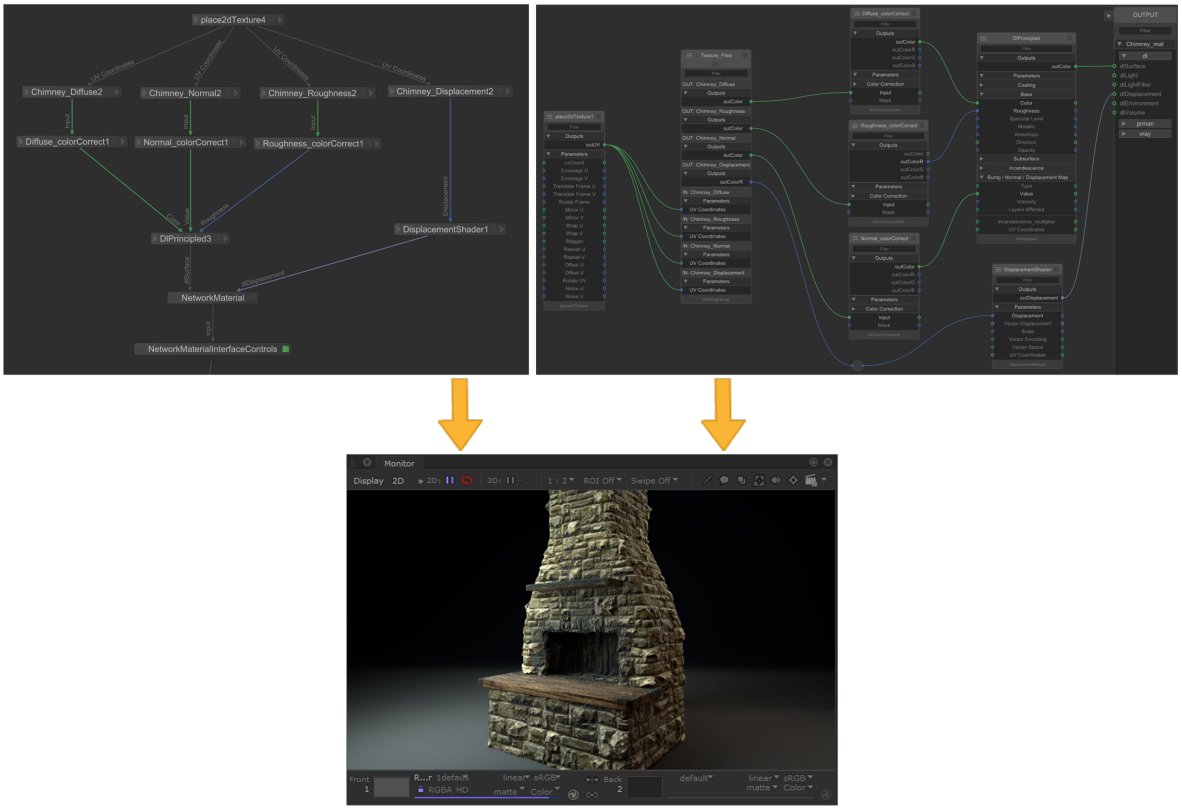
Creating Shading Networks

This topic explains how to use the NetworkMaterialCreate node to build a shading node network.



Note: To learn about the NetworkMaterialCreate node parameters, see NetworkMaterialCreate.

This example illustrates the difference between the previous, and new workflows for building materials. Both networks function in the same way and have the same end result.



Previous NetworkMaterial workflow VS current NetworkMaterialCreate workflow

Rendered result for both workflows

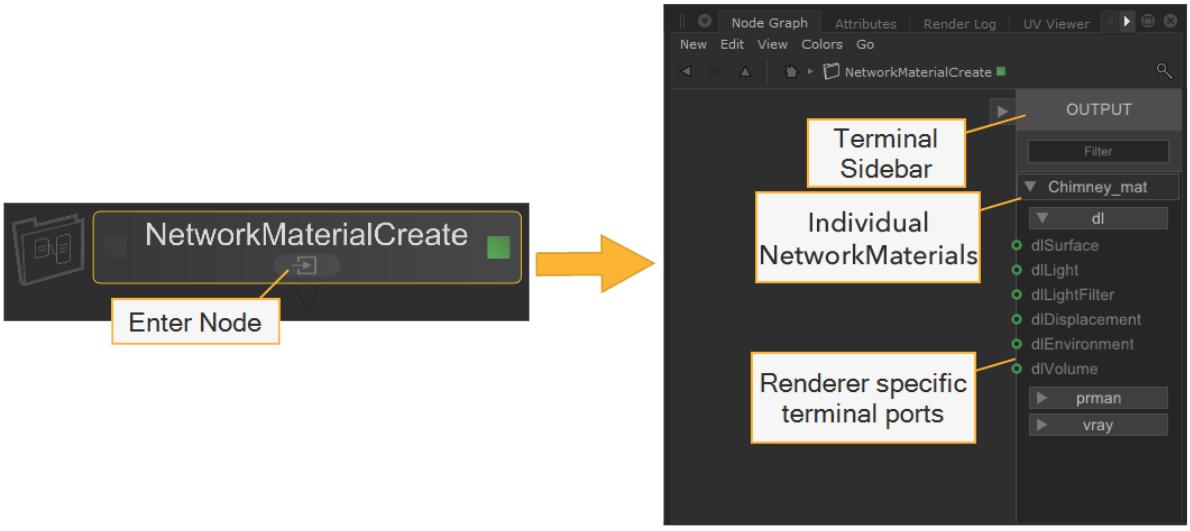
NetworkMaterialCreate Overview

The NetworkMaterialCreate node is created in the same way as any other node, hit **Tab**, select it from the menu and place it in your node graph. To jump inside the node, you can:

- **Ctrl** + **Middle-mouse** click on the node.
- Click the enter node  button.
- Select the node and hit **Ctrl** + **Enter**.

Inside the NetworkMaterialCreate node is a fixed sidebar on the right, which shows each NetworkMaterial and their terminals for each renderer you have set up with Katana. This sidebar functions in the same way as the Network Material node in the previous workflow, but all the terminals are prepopulated so the user doesn't need to add them manually. The sidebar can be hidden and exposed using the collapse  and expand  tabs.

Note: By default, a NetworkMaterialCreate node provides one NetworkMaterial location. If you would like to discover how to set up multiple NetworkMaterial locations within one NetworkMaterialCreate node, see Multiple NetworkMaterials with NetworkMaterialCreate.

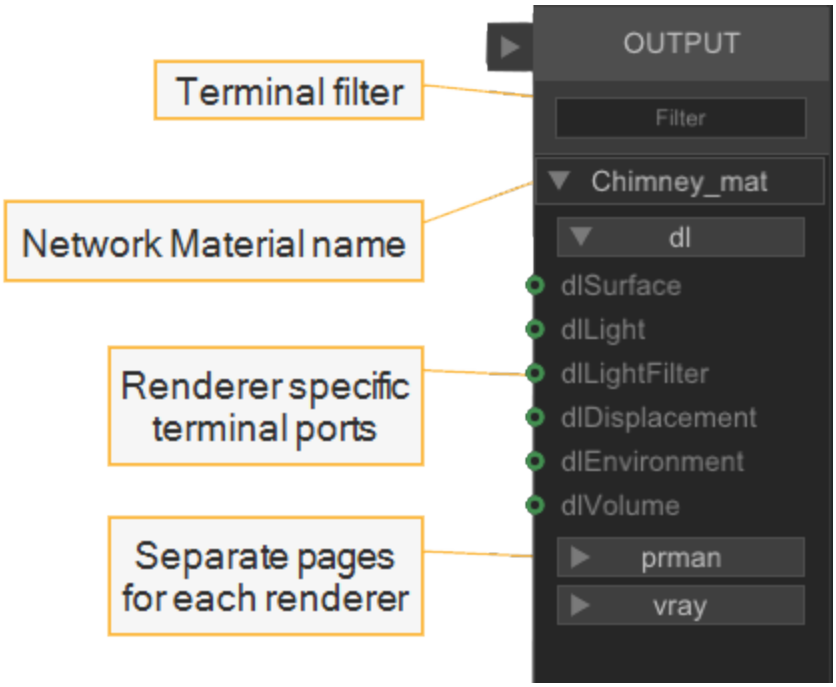


The NetworkMaterialCreate node

Inside the NetworkMaterialCreate node

At the top of this bar there is a **Filter** which allows you to enter a string to search for a specific terminal. For example, start typing 'displacement' and the list filters down to only show the terminals containing the word displacement. This is useful especially if you have multiple renderers or NetworkMaterials set up.

Under the **Filter** bar is the name of your Network Material, you can rename this from the **Parameters** tab, by double clicking your Network Material or selecting it and pressing Enter on the keyboard.



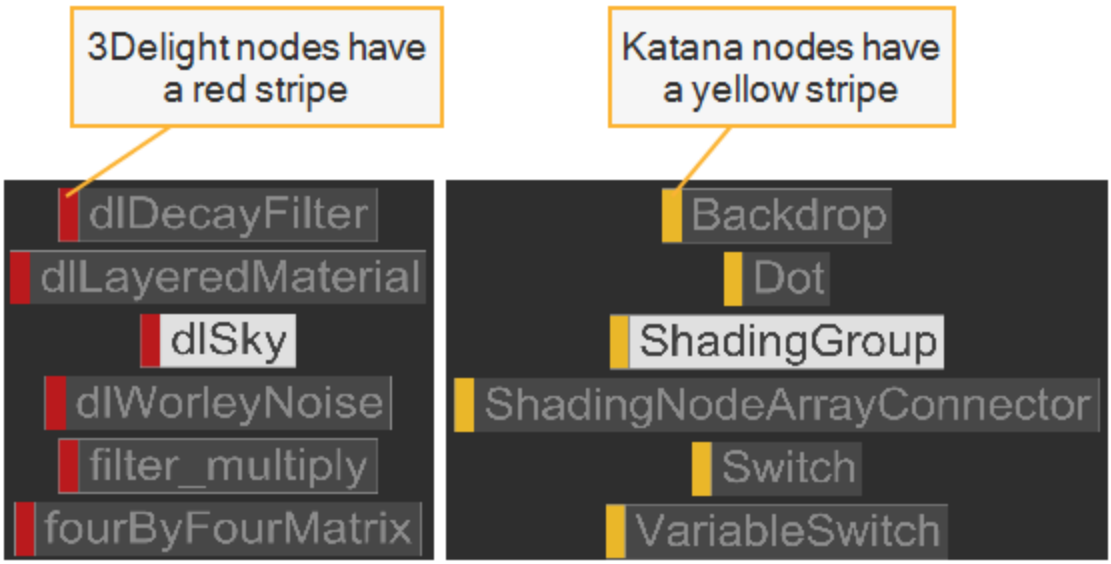
The fixed terminal sidebar

Note: Renaming the node itself won't change the name you'll see on the terminal sidebar. The Network Material name can be changed from the **Parameters** > **Node Parameters** tab.

The Node Menus

To create a shading node, press **Tab** from inside a NetworkMaterialCreate node to bring up the node creation menu, and type the node name. As you're typing, the menu will filter down. When inside a NetworkMaterialCreate node, the menu is limited to show your default renderer nodes, and a few standard Katana nodes, only the nodes you are able to use are visible.

The nodes have a colored stripe on the left to indicate whether they belong to a renderer or whether they are standard Katana nodes. For example, the 3Delight shading nodes are color-coded with a red stripe and the Katana nodes appear yellow.



Colored stripes in the node creation menu indicate group type

You can alter the node menu to display shading nodes from your other renderers. To do this:

1. Hold **Shift** and hit **Tab**.
2. Select the required renderer.
3. Hit **Tab** again to bring up the node menu for your selected renderer.

The **S** key is a 3Delight keyboard shortcut which brings up the node creation menu for 3Delight shading nodes. This can be useful if you want to switch back and forth between two renders as you can use the **S** key to bring up 3Delight shading nodes whilst using **Tab** to bring up the node menu for a different renderer.

Tab	Node menu for selected renderer.
S	Node menu for 3Delight shading nodes.
Shift+Tab	Select renderer to change node menu.

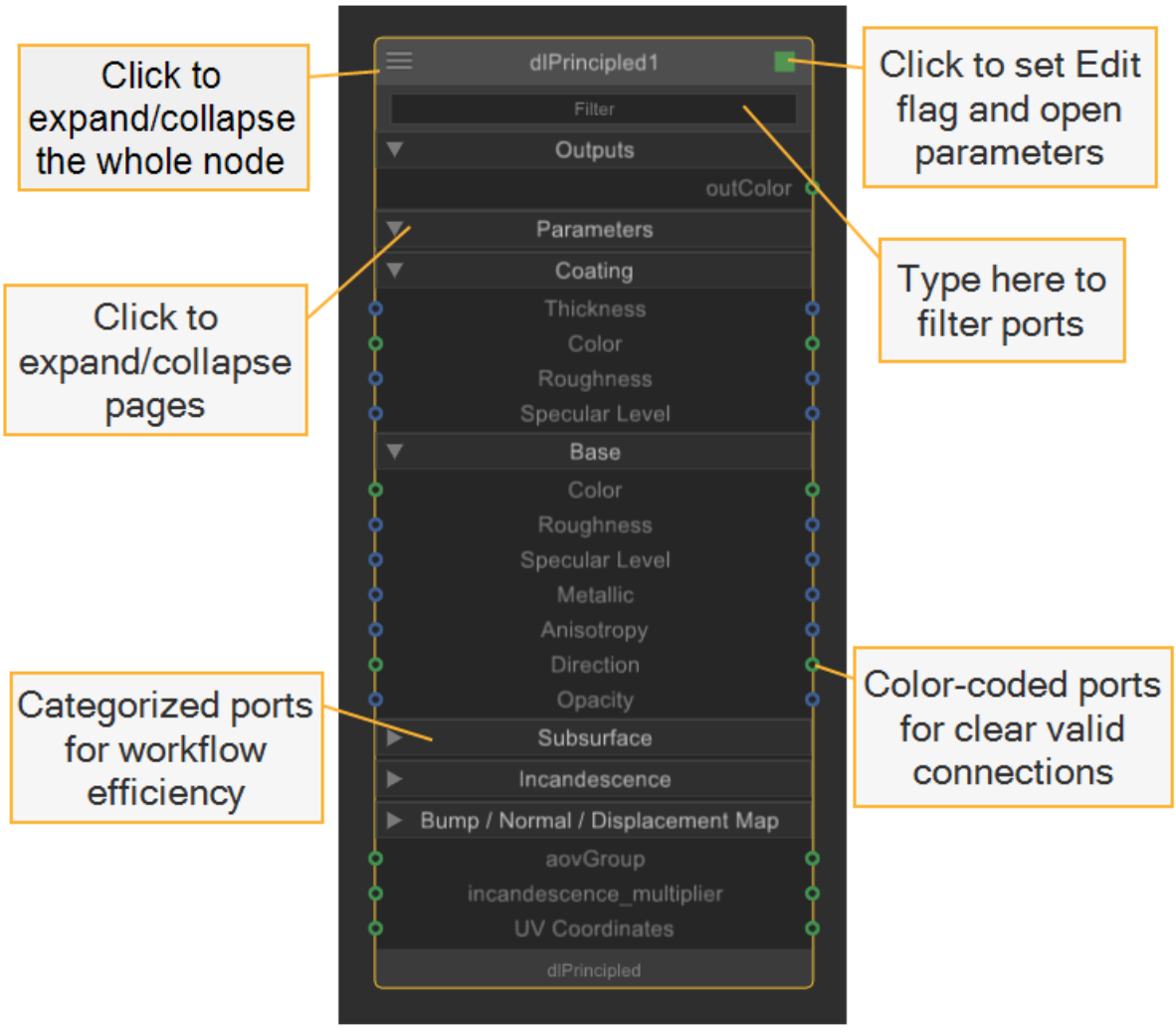
Note: It is possible to change the nodeType from within a shading node's parameters. Values of any parameter names that overlap between node types will be remembered and no changes will be lost if you want to switch back and forth.

Connecting Shading Nodes

Setting up shading nodes within the NetworkMaterialCreate node is designed to make things as simple as possible for artists. Shading networks set up inside the group feature a left-to-right workflow which is well suited to working with a large number of shading nodes.

The shading nodes themselves are designed and optimized for creating materials, as the input and output ports are all visible and clearly labeled.

Tip: You can rename the shading nodes by selecting the node, hitting **Enter**, and typing the new name. You can also show and hide the **Filter** function on selected nodes using the **Alt**+**Enter** keyboard shortcut.



Shading Node UI

Tip: Some pairs or groups of nodes that are normally used together or that rely on each other, are automatically created when you place one of the nodes. For example, if you place a **file** node, a **place2DTexture** node is automatically created.

To link shading nodes together, click once on a parameter port to begin drawing the connection, and then click once on the other shading node's port to connect the two nodes together. Invalid target ports are grayed out and disabled to show which connections are available.

The input and output ports are all color-coded to indicate which connections can be made. You can only connect the compatible data types, for example, int to int and float to float.

Tip: Hover your cursor over the input/ouput port to see what data type it provides/receives.

Tip: Hover over a connection and press '**/**' on your keyboard to follow the connection to its connected node.

Data Type Color Codes:

- color
- float / array_float
- int
- matrix
- normal
- point
- string
- vector
- disabled

misc

Once your shading nodes are connected to the terminal sidebar, the network is now set up and your NetworkMaterial is in your **Scene Graph** under /root/materials by default.

Note: To learn how to change your NetworkMaterial scene graph location and other NetworkMaterialCreate parameters, see NetworkMaterialCreate.

Tip: In the same way as the terminal sidebar, you can type in the **Filter** field of the shading nodes to quickly search for an input/output, even if the menu is not expanded. You can show and hide the **Filter** function on selected nodes using the **Alt**+**Enter** keyboard shortcut.

The arrows on the shading nodes show if a page is expanded:

- ▼ - A downwards-facing arrow means the page is expanded.
- ▶ - A right-facing arrow means the page is collapsed.

You can click on the page titles to expand/collapse pages so that you can hide certain sections. To collapse/expand the entire shading node, you can click the expand/collapse state button ☰ at the top-left of a node or use the keyboard shortcuts:

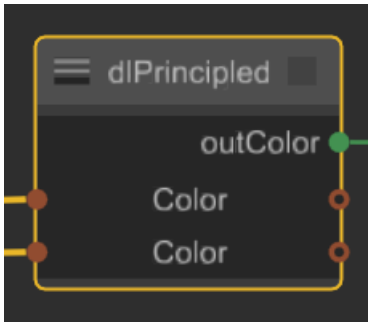
Tip: You can use the preference **nodegraph** > **defaultShadingNodeViewState** to set the default expand/collpase state of new nodes.

- **ALT + 1** - Collapse completely.

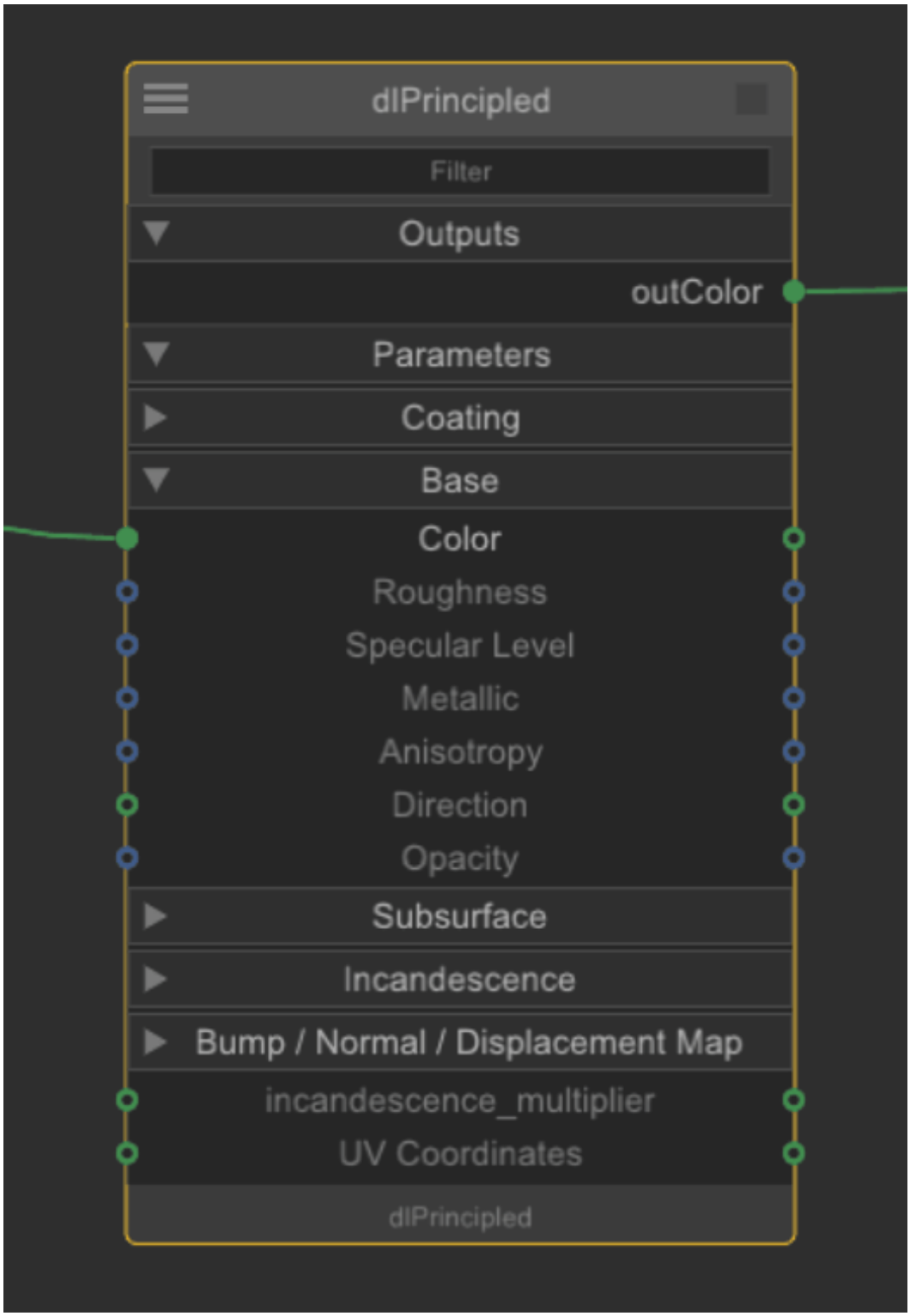


- **ALT + 2** - Expand to show connected ports.

Tip: The example shows just connections exposed, but you can expose the pages containing the connected inputs and outputs, such as **Base** by enabling the **showPagesConnectedOnly** control in the **Preferences** under **nodegraph**.

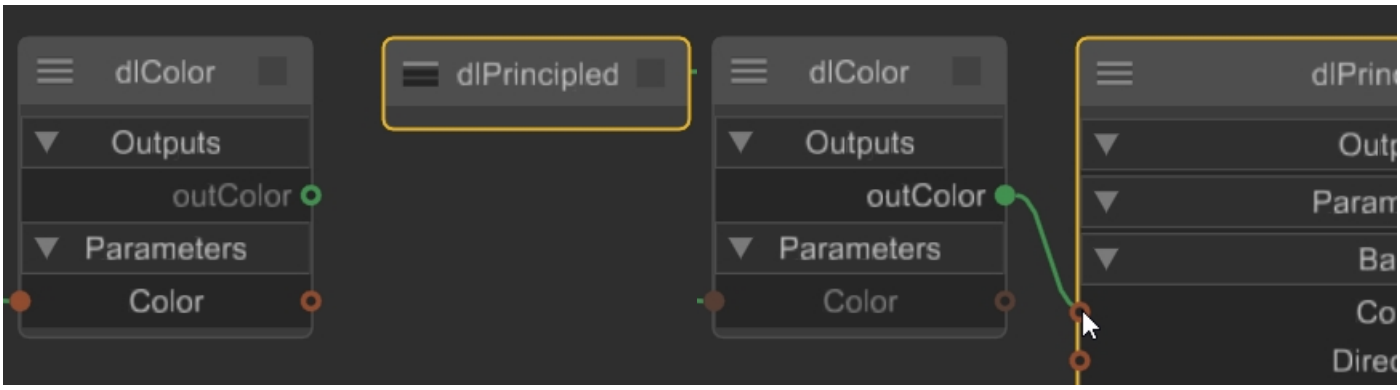


- **ALT + 3** - Fully expand pages and connections.



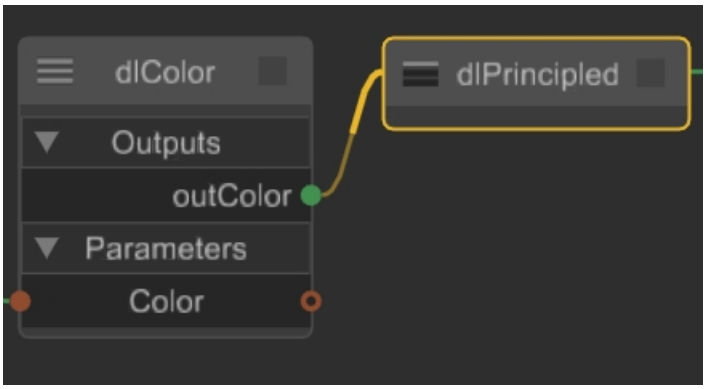
When collapsed, you can hover over a shading node while pressing and holding the **X** key and it will temporarily expand. It will collapse when released.

If a node is collapsed, you can also drag a connection over the node to automatically expose compatible connections. The node collapses again automatically after connecting the input or output. The same is true if you change you mind and drop the connection outside the node.



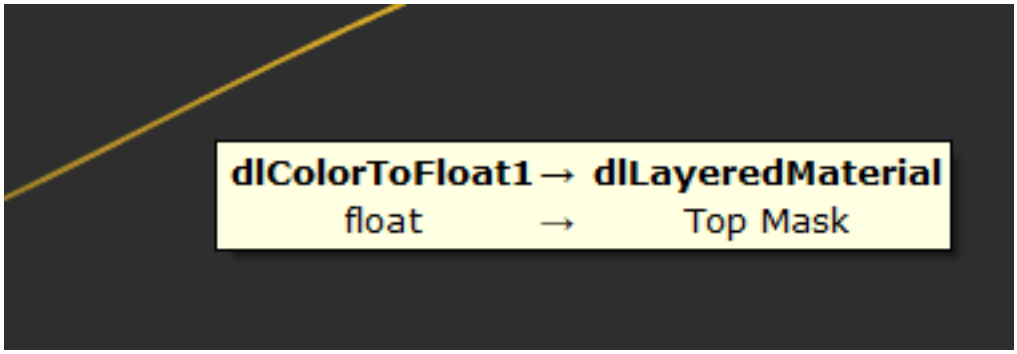
A collapsed dlPrincipled target node with no exposed connections

Dragging a connection over the node auto-expands compatible connections



The auto-collapsed node after a connection is made

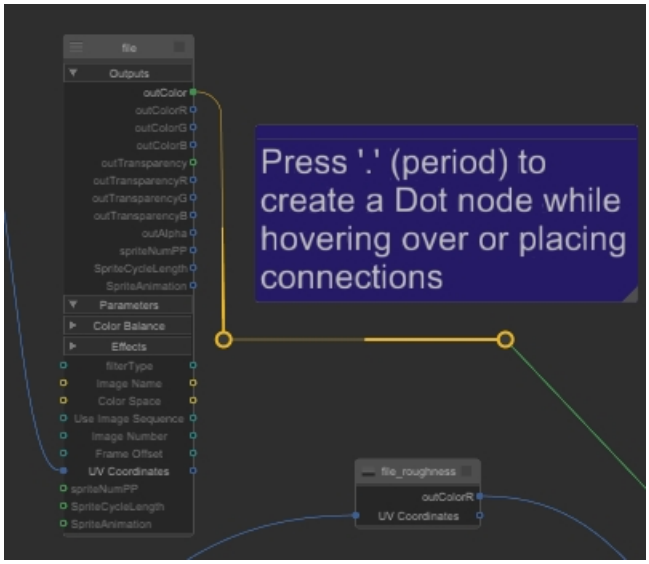
You can hover over a connection to display a tooltip detailing the node the connection originates from and the node it is connected to downstream.



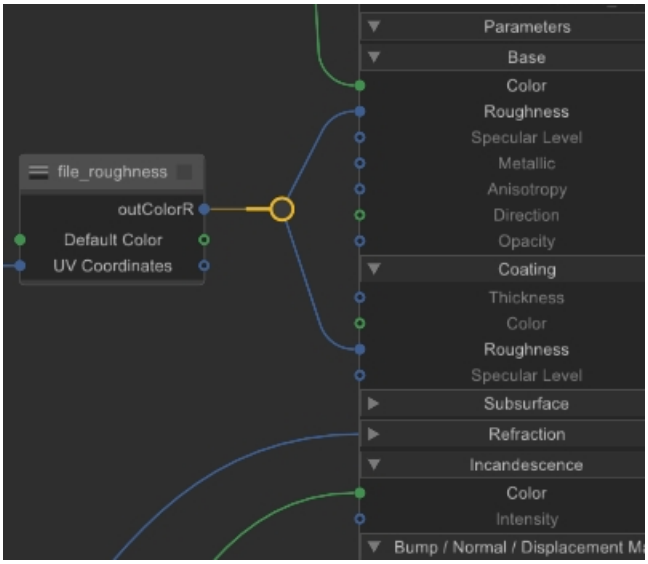
The data types of each connection are also shown.

Organizing a Shading Network with Dot Nodes

Dot nodes can help you organize complex shading networks to make them easier to read. For example, you can bend connections around notes for other artists or use a Dot to connect a single output to multiple inputs.



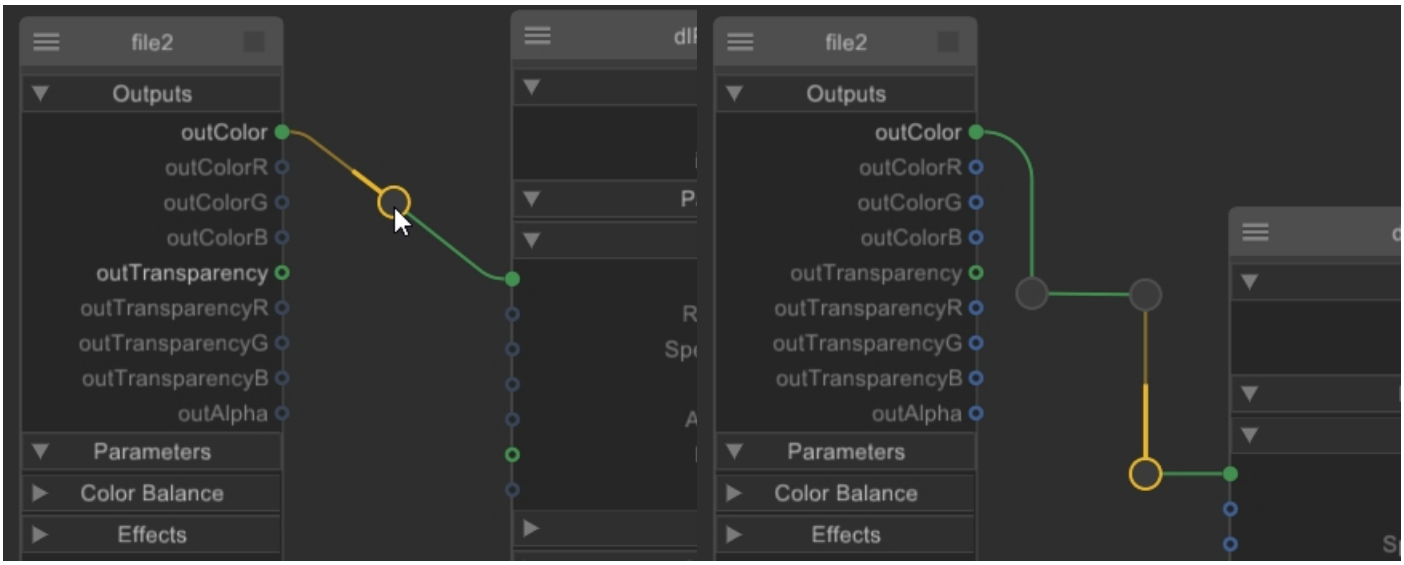
Using Dot nodes to bend connections



Splitting an input using a Dot node

You can add a Dot like any other node, by pressing **Tab** and then typing **Dot** into the node finder, but the fastest way to add a Dot node is to press **.** (period) on your keyboard. Like other nodes, the Dot node sticks to your pointer and you can click anywhere in the shading network to place it.

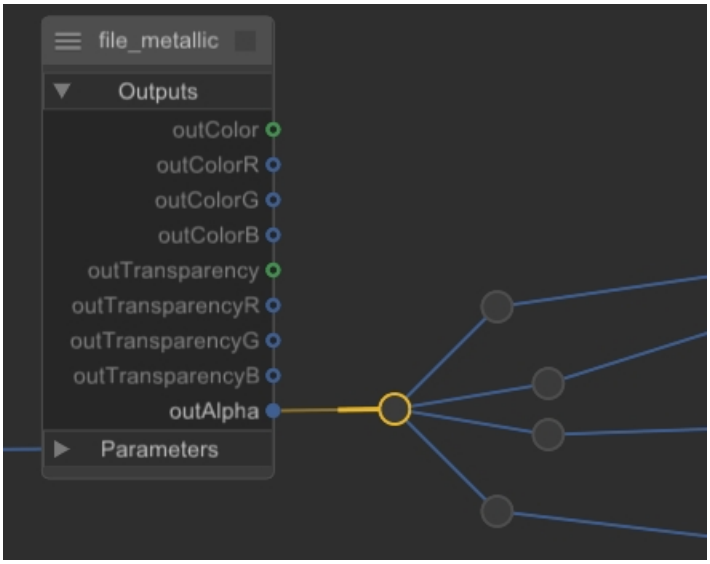
Alternatively, you can add Dot nodes when hovering over a connection to insert the Dot between nodes or you can drag a node output and then add a Dot node to create a Dot chain.



Adding a Dot node to a connection

Adding multiple Dot nodes to create a chain

Dot nodes in shading networks are omnidirectional, meaning you can connect to them and drag outputs from them in any direction. They can have as many output connections as you like, but only one input connection.

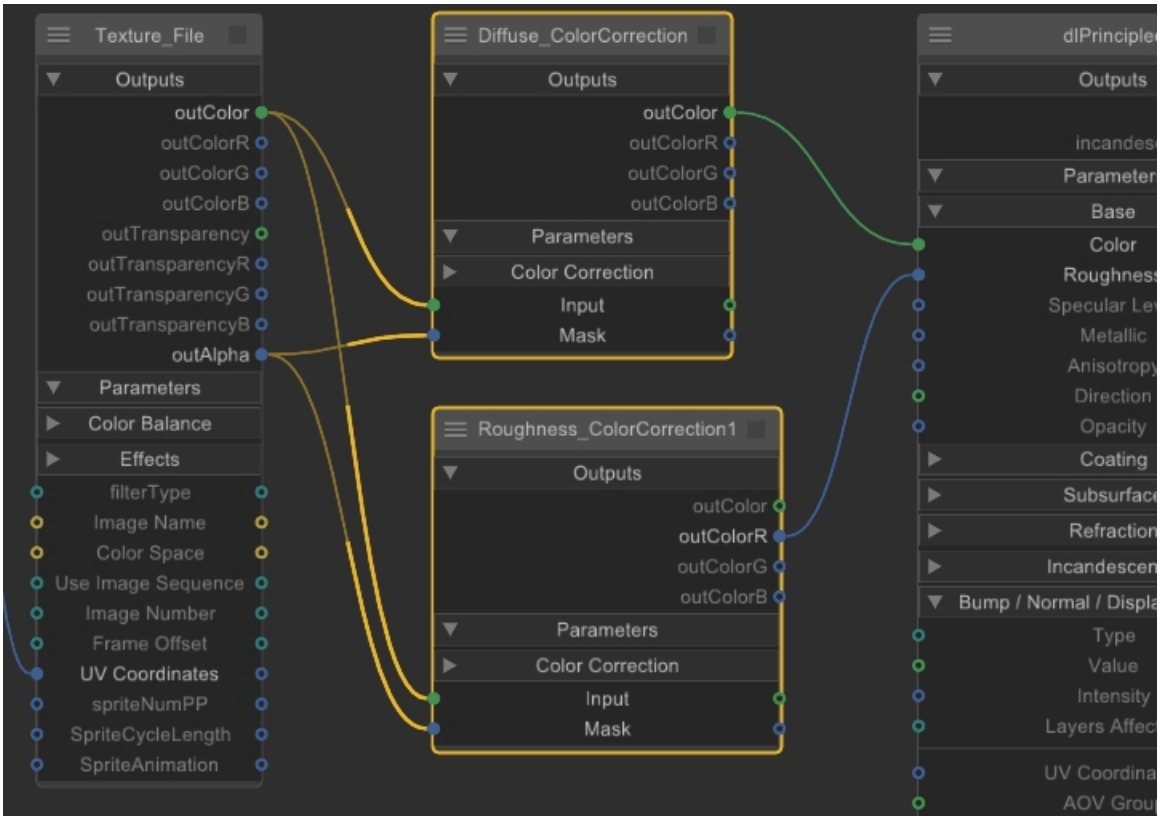


Hiding Node Connections

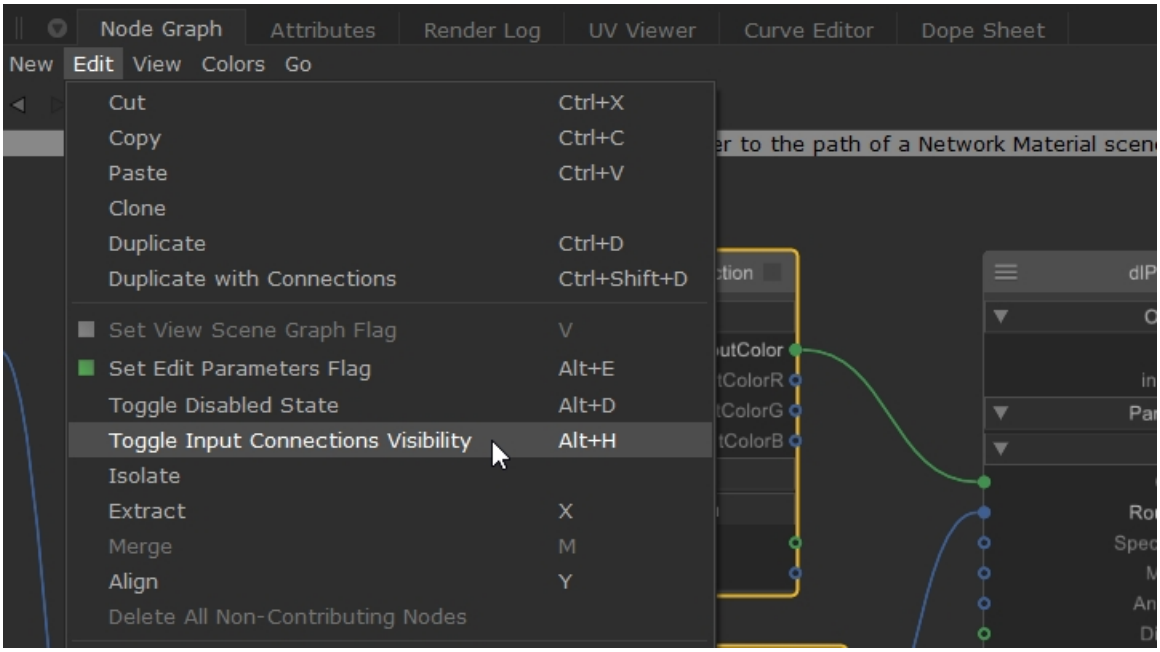
In node-heavy shading networks, the connections between nodes can cause confusion if you're looking for a particular input. You can show and hide node input connections to clean up the network by navigating to **Edit** > **Toggle Input Connection Visibility** or by using the **Alt**+**H** keyboard shortcut.

Tip: You can also show and hide the **Filter** function on selected nodes using the **Alt**+**Enter** keyboard shortcut.

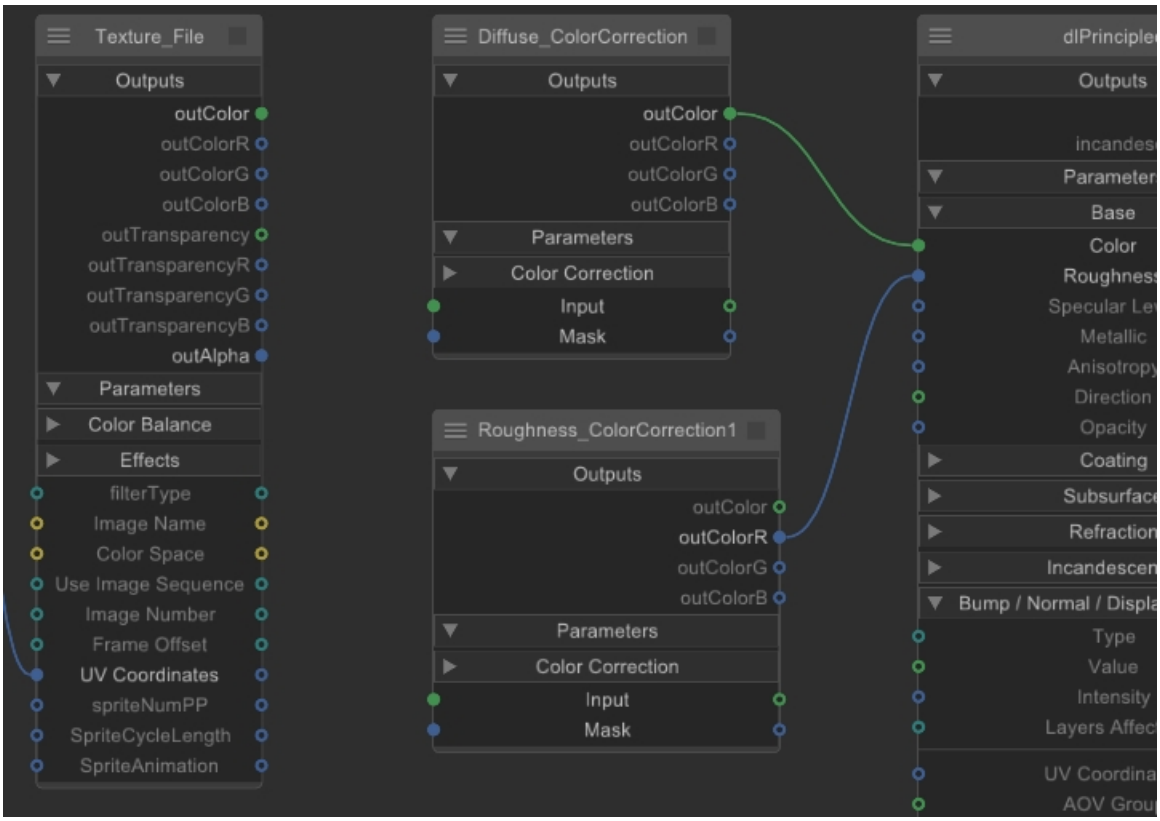
1. Select the node or nodes you want to affect in the shader tree. In this example, the Diffuse and Roughness nodes.



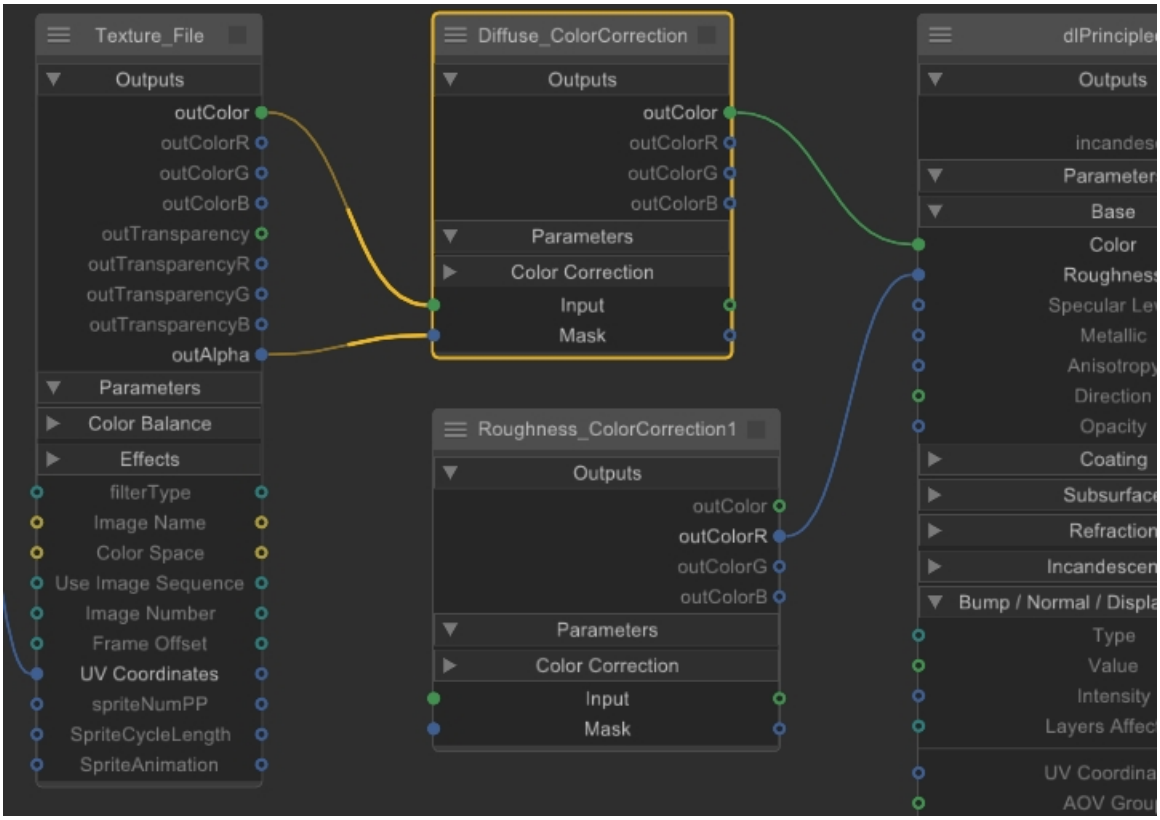
2. Navigate to **Edit** > **Toggle Input Connection Visibility** or use the **Alt**+**H** keyboard shortcut to hide the input connections on the selected nodes.



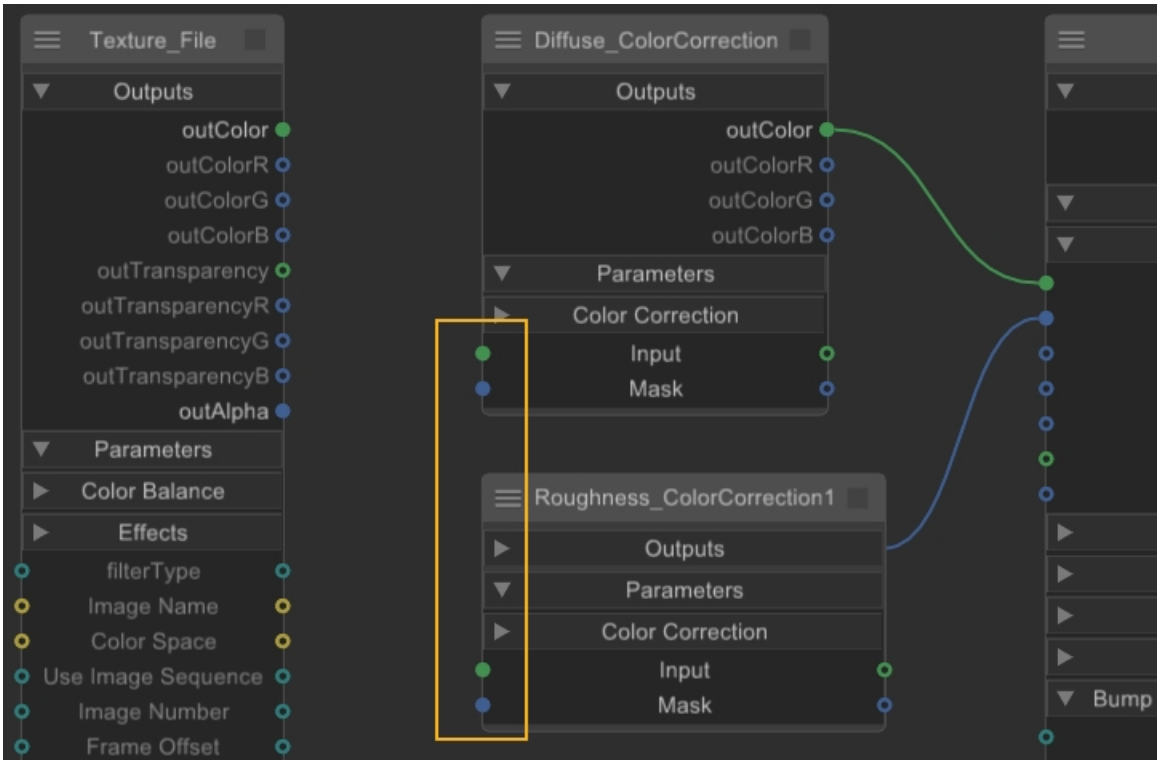
3. Deselect the nodes to see the effect.



4. You can select a node in the tree to temporarily display its input connections.



5. When the inputs are hidden, the connections on the node are still filled to indicate that a hidden connection exists.



6. Select the nodes and navigate to **Edit** > **Toggle Input Connection Visibility** or use the **Alt**+**H** keyboard shortcut to show the input connections on the selected nodes.

7. **Tip:** Hold **Alt**+**H** with no selection to temporarily show hidden connections.

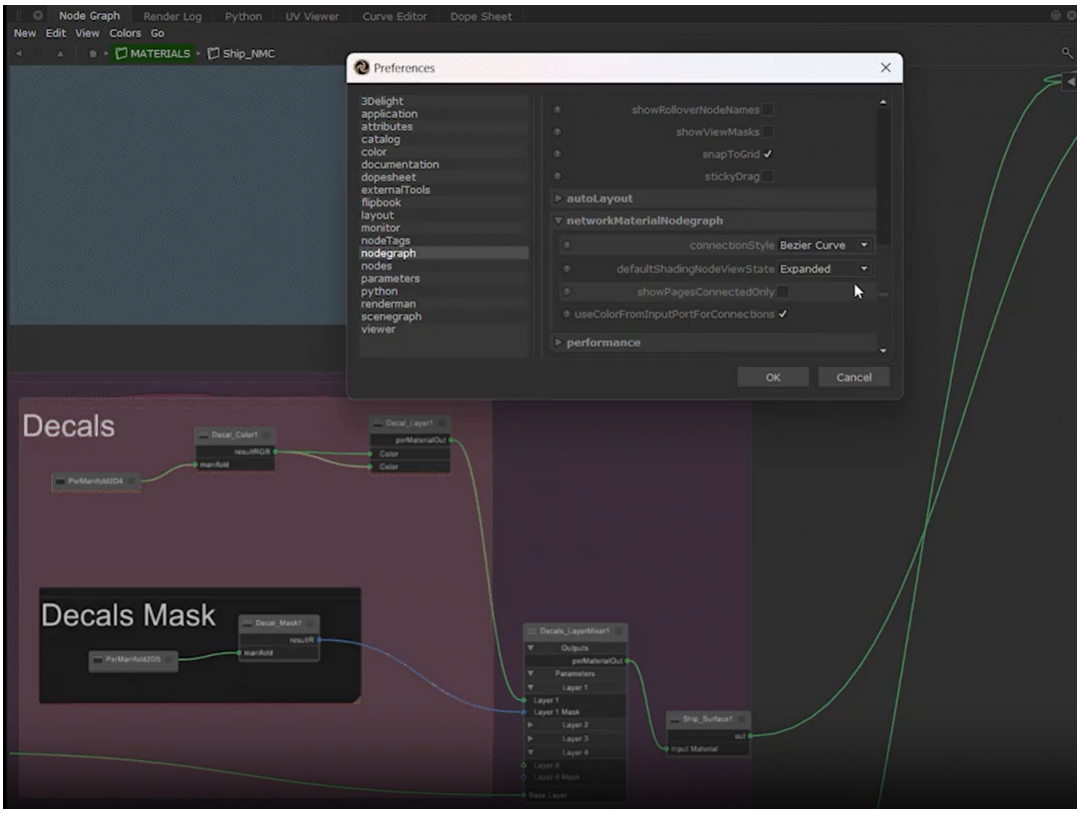
Customizing Drawing Style for Node Connections

1. Go into **Edit > Preferences > nodegraph**

2. Under the **networkMaterialNodegraph** page, modify the **connectionStyle** preference. There are three options:

    - **Line** - all connections are straight
    - **Short Curve** - This will enforce a curve at the point of which it meets the port, similar to the one used when connections are made against a Dot node in this context
    - **Bezier Curve** - This is the default for the material context, connections are drawn as bezier curves

For the above, these options will be set within the layout preferences and not saved as part of the materials created.



Multiple NetworkMaterials with NetworkMaterialCreate

You can create multiple NetworkMaterials within one NetworkMaterialCreate node. This allows you to share shading nodes across different network materials, reducing the number of nodes needed when creating variants of a material.

The **Material Scenegraph** within the NetworkMaterialCreate **Parameters** allows you to organize your NetworkMaterials through rearranging and adding Namespaces. Adding a Namespace in your Material Scenegraph creates a group in your Scene Graph tab, which can parent multiple Network Materials and other Namespaces for the purpose of organization.

Note: For more information about the Material Scenegraph, refer to the Organizing NetworkMaterials and Namespaces section of this topic.
For more information about the NetworkMaterialCreate Parameters, see NetworkMaterialCreate.

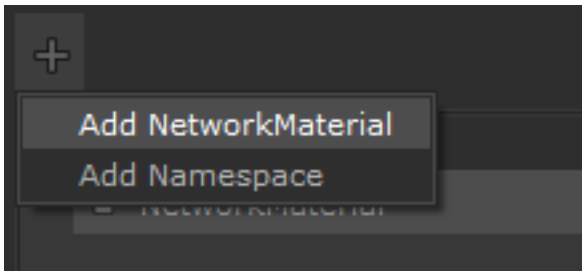
Adding a New NetworkMaterial

To add a new NetworkMaterial to a NetworkMaterialCreate node:

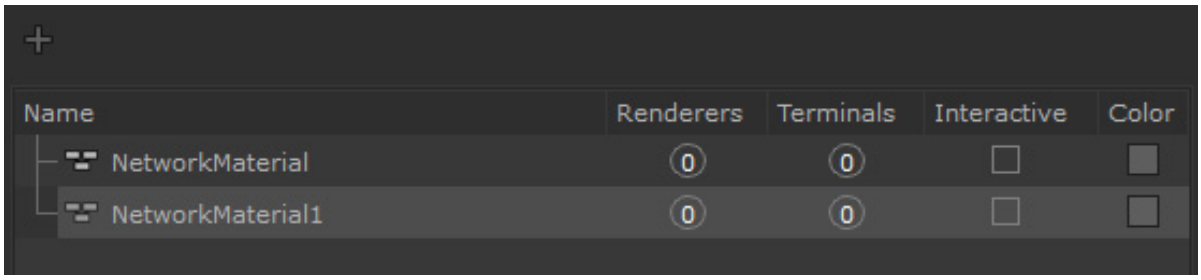
1. Open the NetworkMaterialCreate node **Parameters** by activating the edit flag.



2. Click the plus button and select **Add NetworkMaterial**.

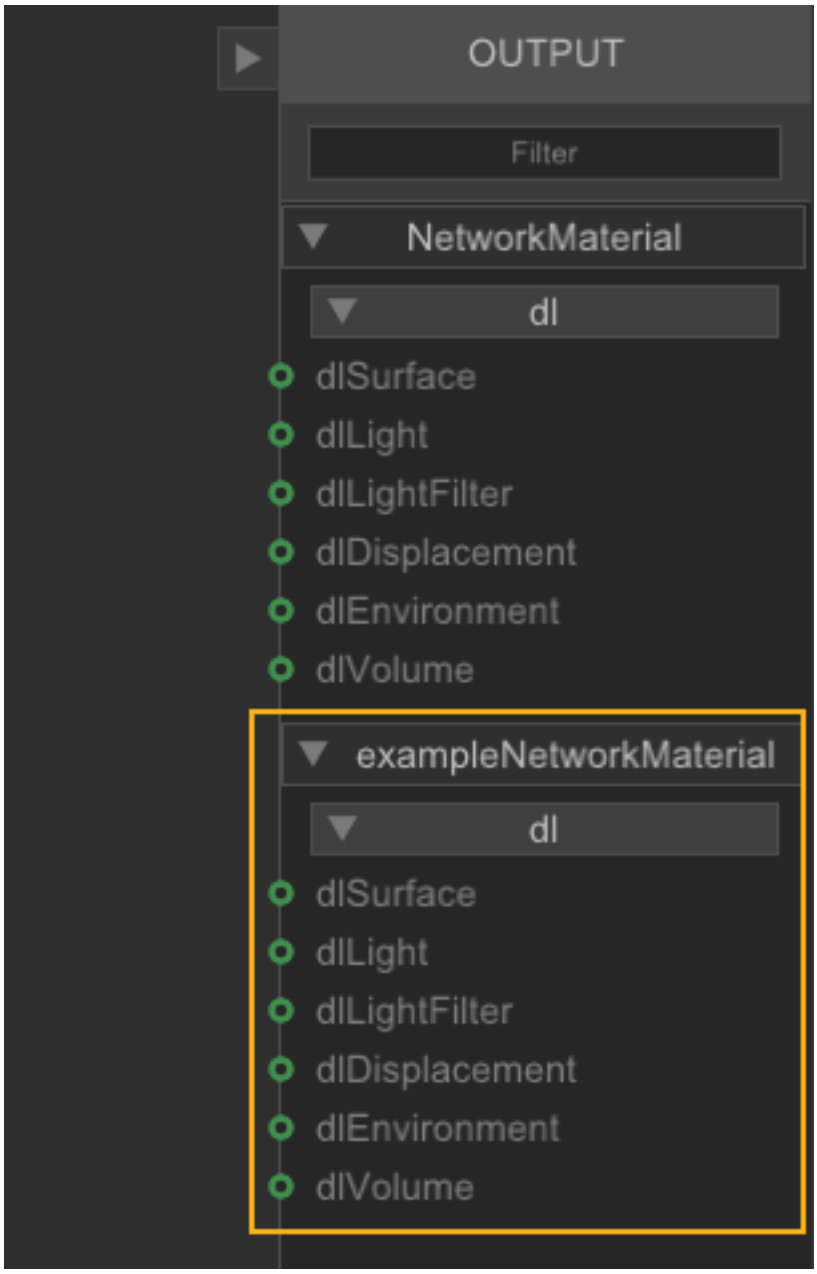


A new NetworkMaterial is created.



New NetworkMaterial in the Material Scenegraph

3. Change the name of the new NetworkMaterial by selecting it and pressing **Enter** on the keyboard, or by double-clicking.



New NetworkMaterial in the terminal sidebar

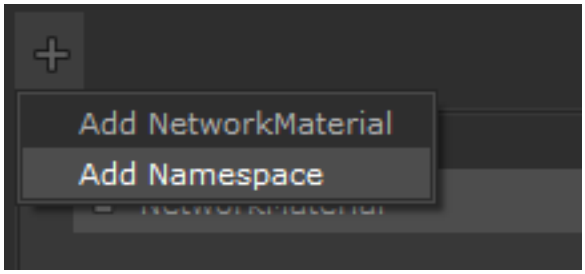
Adding a New Namespace

To add a new Namespace to a NetworkMaterialCreate node:

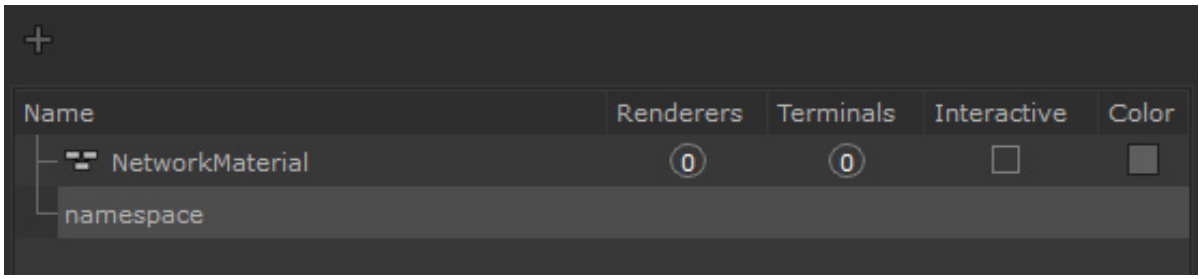
1. Open the NetworkMaterialCreate node **Parameters** by activating the edit flag.



2. Click the plus button and select **Add Namespace**.

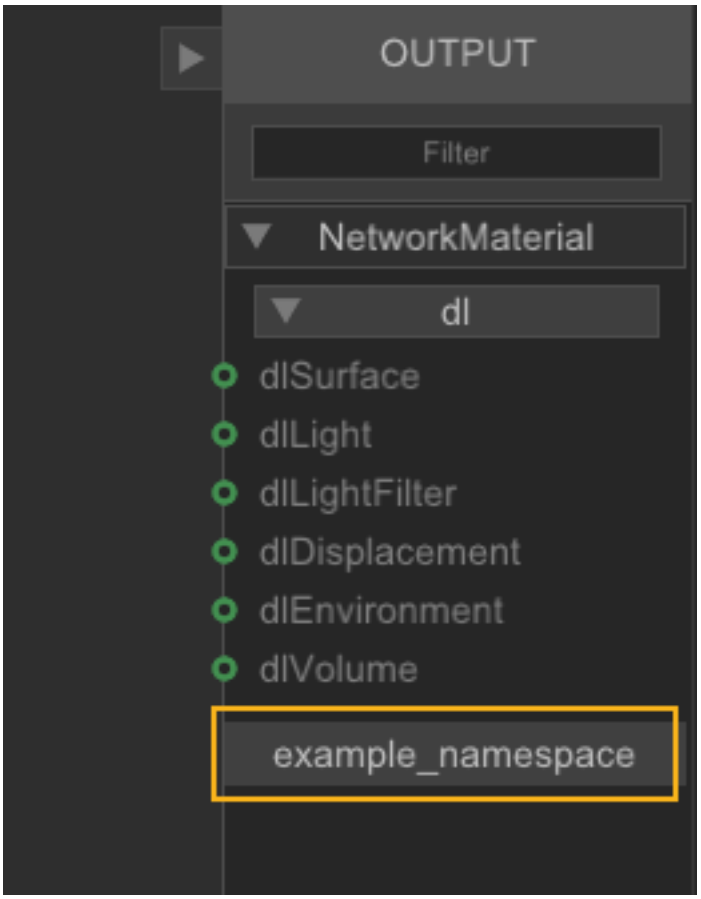


A new Namespace is created.



New Namespace in the Material Scenegraph

3. Change the name of the new Namespace by selecting it and pressing **Enter** on the keyboard, or by double-clicking.



New Namespace in the terminal sidebar

Note: You can also add NetworkMaterials and Namespaces through the menu options when right-clicking within the NetworkMaterialCreate Material Scenegraph.

Organizing NetworkMaterials and Namespaces

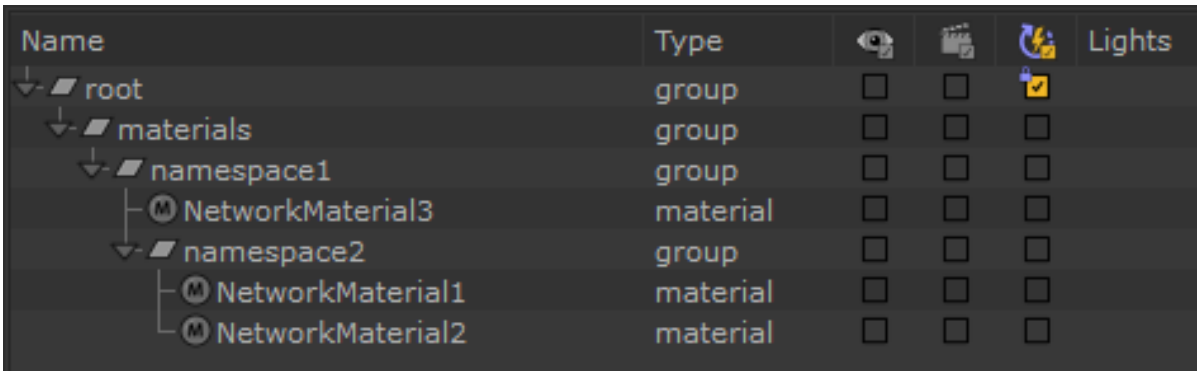
You can organize your NetworkMaterials and Namespaces through the NetworkMaterialCreate Material Scenegraph in node **Parameters**.

- Use the middle-mouse button to click and drag the NetworkMaterials and Namespaces into the structure you require.
  This structure is reflected in the **Scene Graph** and the terminal sidebar inside the NetworkMaterialCreate node.

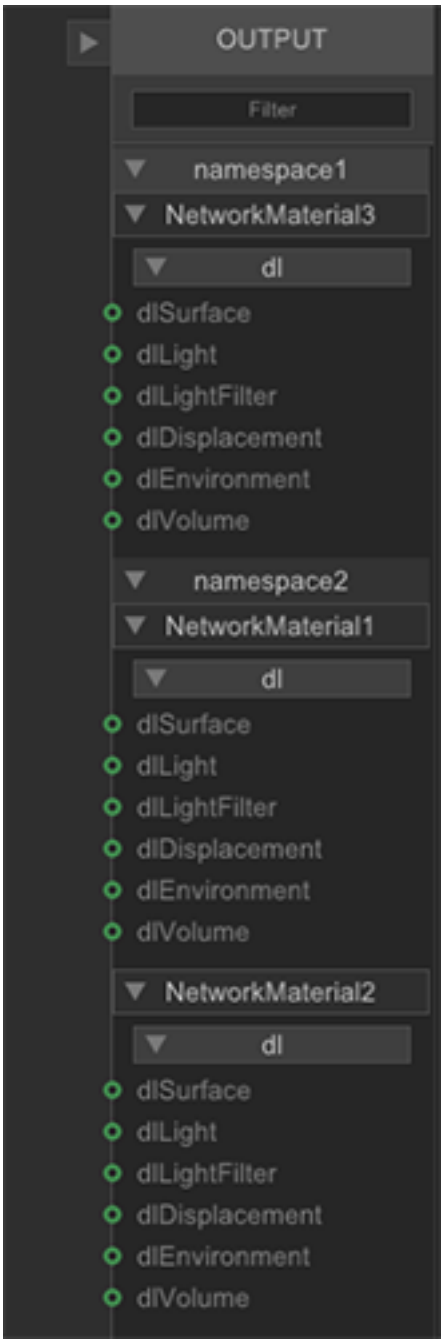
Note: Rick-click in the Material Scenegraph to delete and duplicate NetworkMaterials and Namespaces as well as fully expand or collapse them.



NetworkMaterials and Namespaces organized in the Material Scenegraph



NetworkMaterial and Namespace structure in the **Scene Graph**

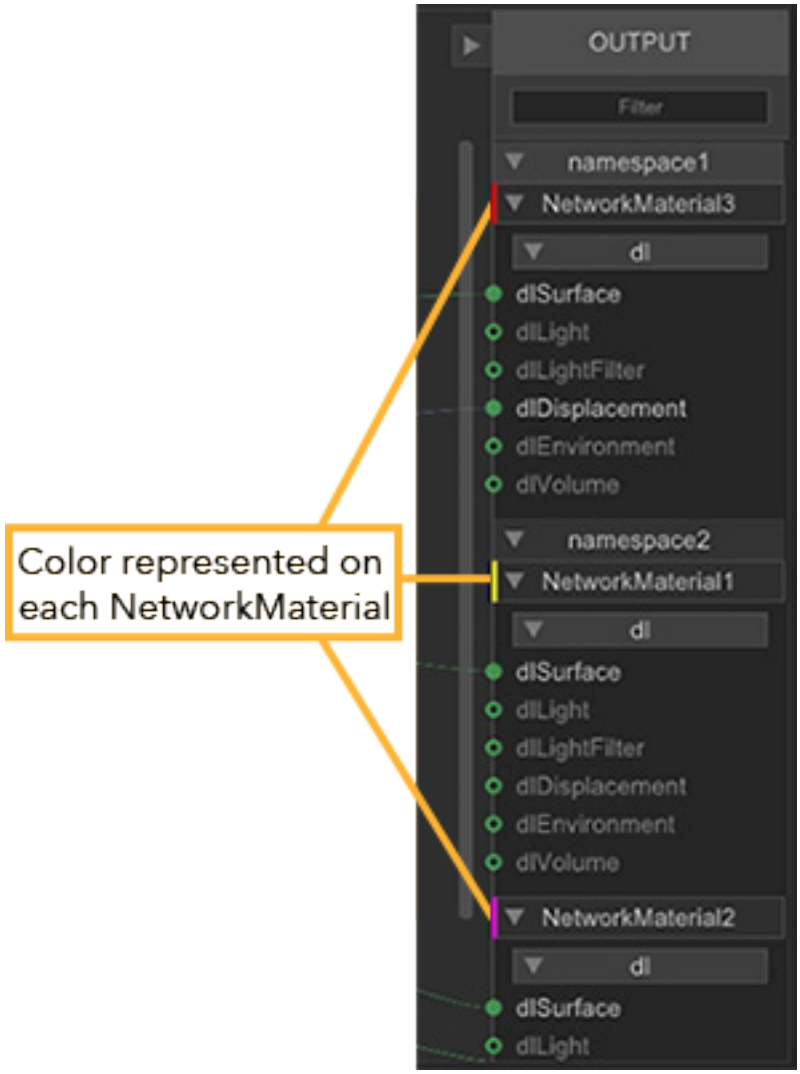


NetworkMaterial and Namespace structure reflected in the terminal sidebar

The Material Scenegraph also provides some information about each NetworkMaterial. You can see how many renderers and terminals are connected to each NetworkMaterial, set the interactive state of a NetworkMaterial, and assign it a color.

Note: For more information about the NetworkMaterialCreate node parameters, see NetworkMaterialCreate.

The color helps to distinguish multiple NetworkMaterials at a glance. These colors are represented on the NetworkMaterial names on the terminal sidebar within the NetworkMaterialCreate node.



The NetworkMaterials and Namespaces can be collapsed and expanded using the arrows on the terminal sidebar.

Workflow Example

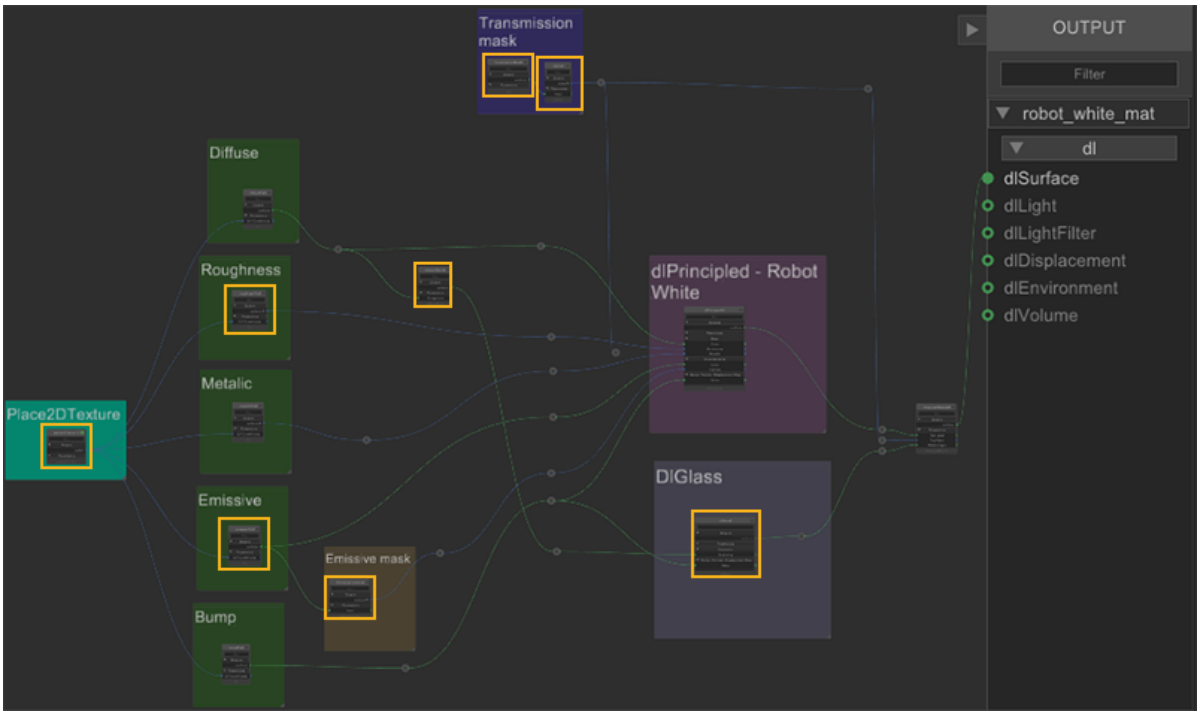
In this example there are two robot characters each requiring a different material variation. The ability to create multiple NetworkMaterials from one NetworkMaterialCreate node is very useful in this situation as

each material variation uses some of the same textures and masks.

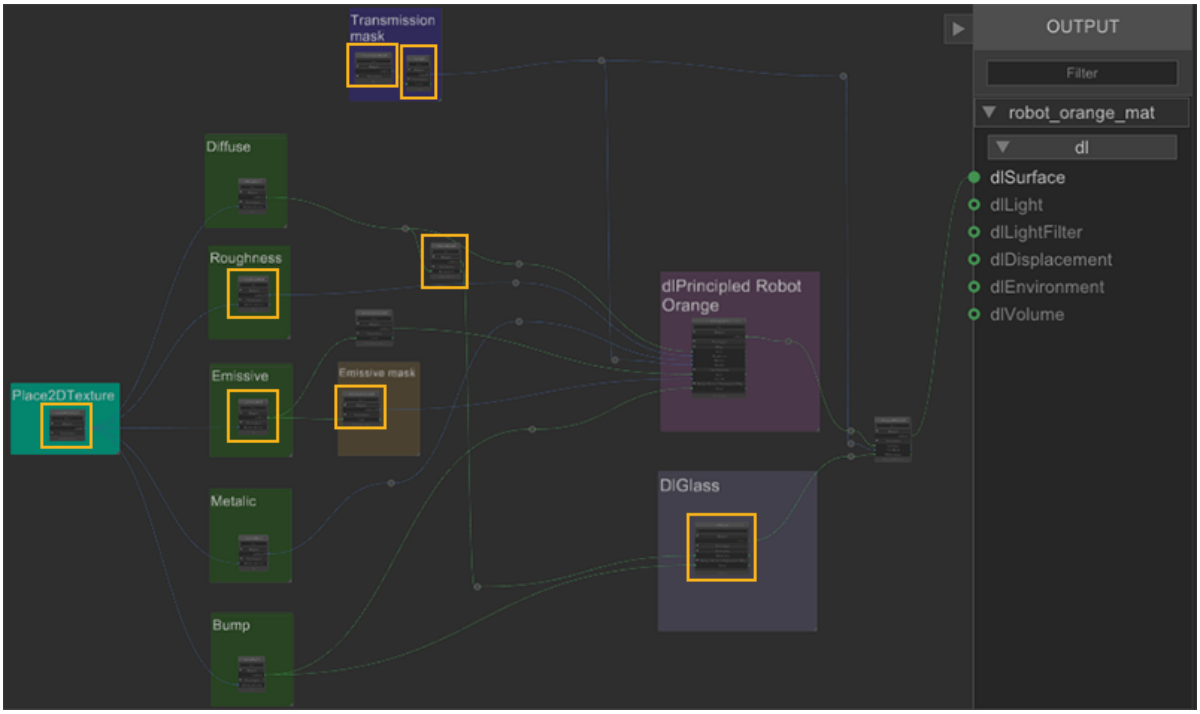


Each of the robots' materials share a transmission mask, roughness and emissive texture files, and require the same glass shader.

Using a separate NetworkMaterialCreate node for each NetworkMaterial would require a lot of duplicated nodes resulting in more nodes overall. This simple example shows how sharing parts of the network reduces duplication. In production scripts featuring hundreds of nodes, the power of multiple NetworkMaterials within a single NetworkMaterialCreate node is far greater.



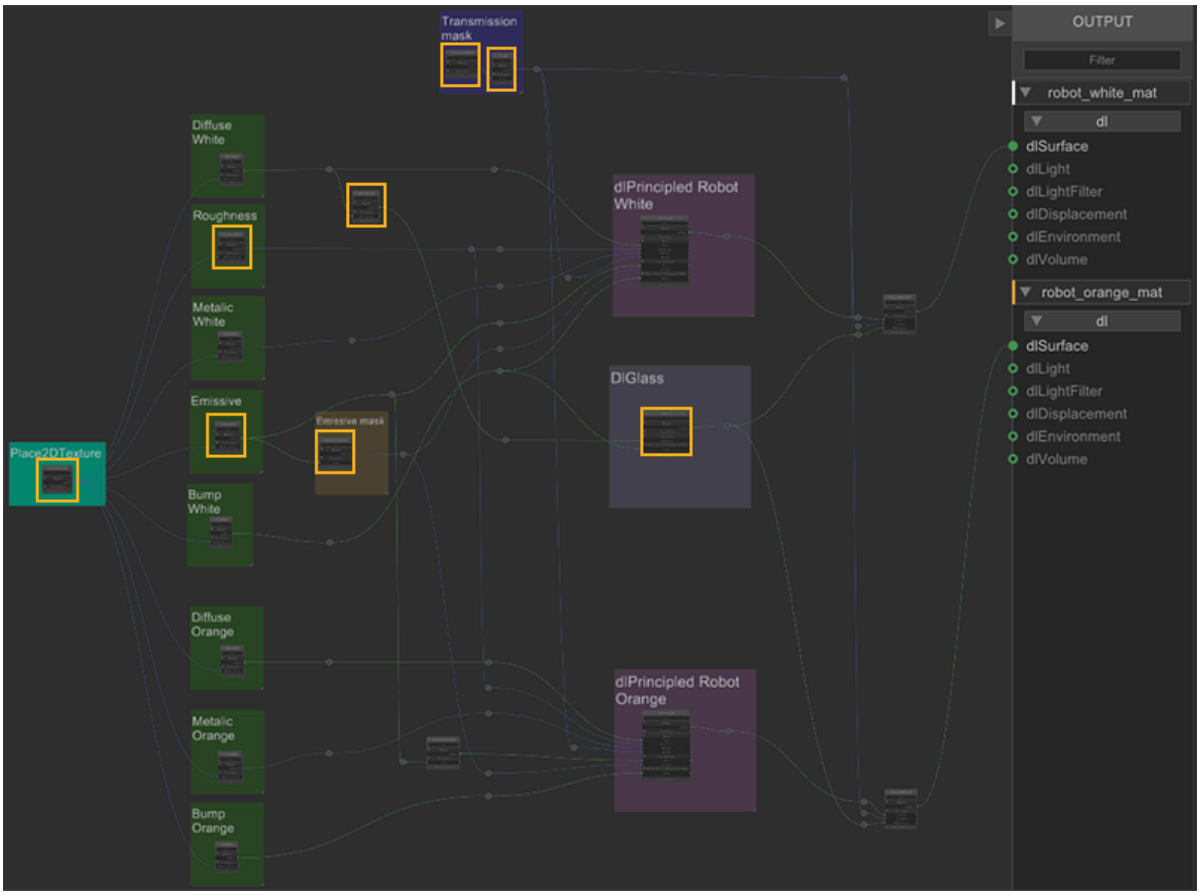
A NetworkMaterialCreate node with one NetworkMaterial location for a white material. Duplicates highlighted.



A NetworkMaterialCreate node with one NetworkMaterial location for an orange material. Duplicates highlighted.

Using one NetworkMaterialCreate node with two NetworkMaterials means that they can share sections of the shading node network, in this case, the 8 highlighted nodes. This reduces the number of nodes from 27

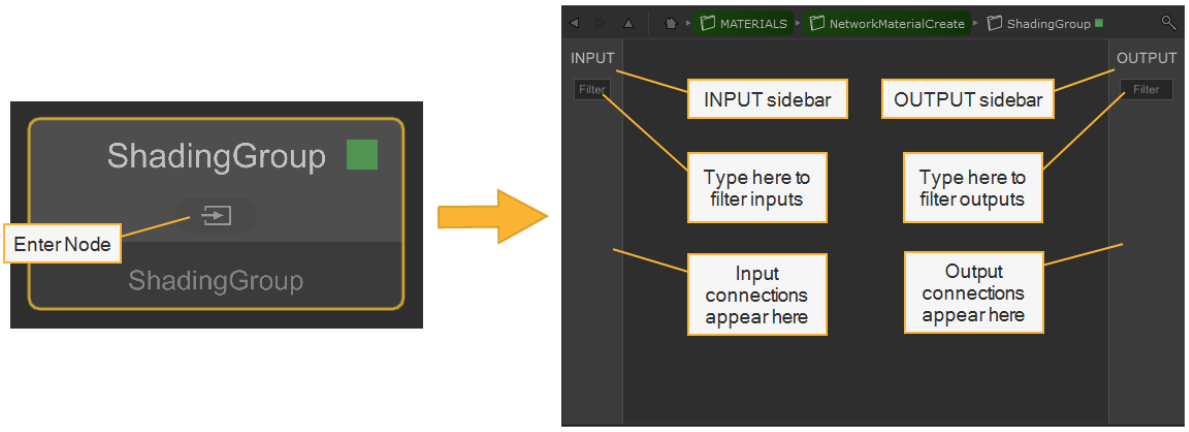
over two separate NetworkMaterialCreate nodes, to 19 nodes in one NetworkMaterialCreate node by sharing 8 nodes.



A NetworkMaterialCreate node with two NetworkMaterial locations. Shared nodes highlighted.

Organizing Shading Networks with ShadingGroup Nodes

The ShadingGroup node is designed to keep your workspace organized by allowing you to group sections of your shading node network together. Inside a ShadingGroup node, there are fixed input and output bars, which are used to connect the nodes within the group to the rest of the network. This also means you are able to view and access the input and output ports of the nodes within the group on the exterior of the ShadingGroup node. These features of ShadingGroup nodes help to provide the artist with a user friendly interface and enables you to hide any unnecessary details from your shading node network.



The ShadingGroup node

Note: The ShadingGroup node is designed to be used within the NetworkMaterialCreate node and can therefore only be created whilst inside the NetworkMaterialCreate node.

Using the ShadingGroup Node

To create a ShadingGroup:

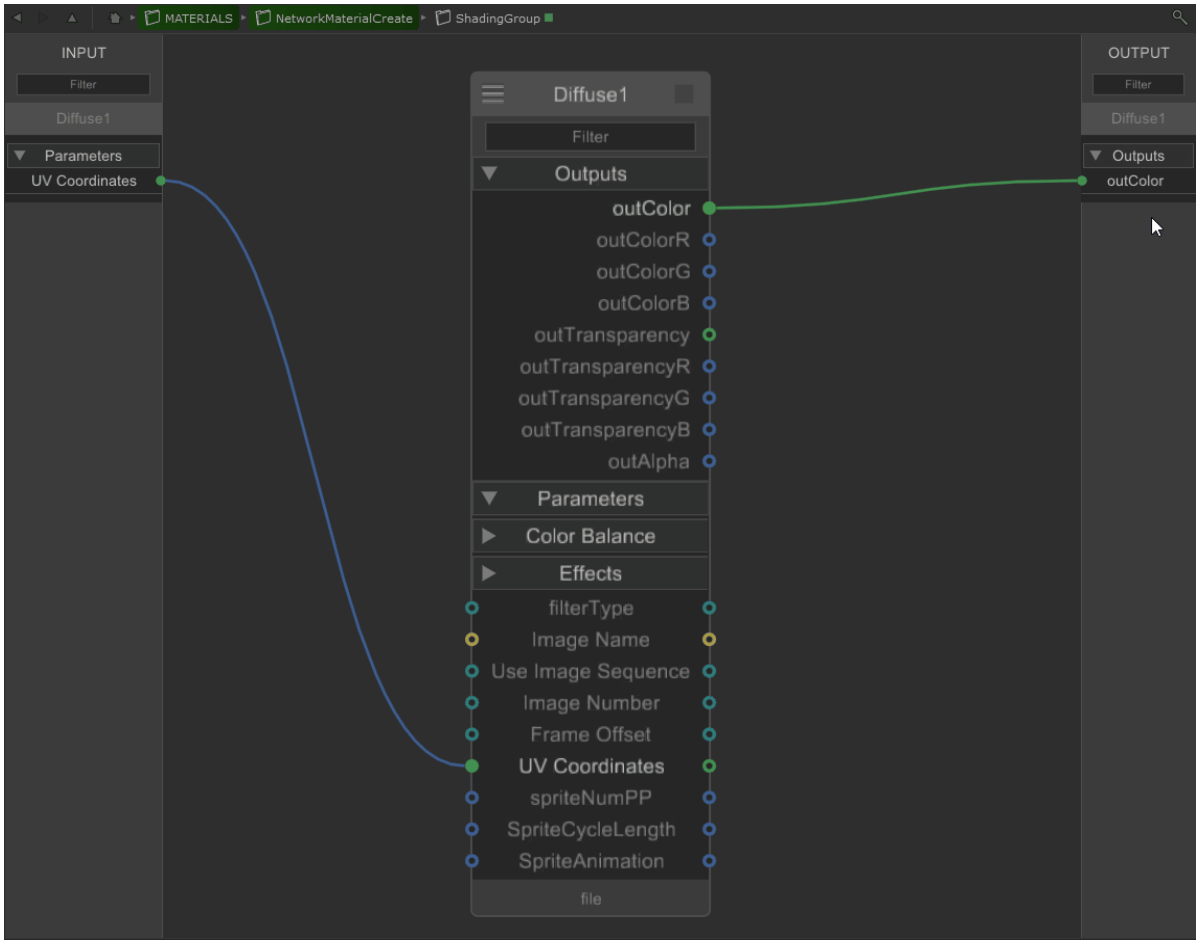
1. Hit **Tab**.
2. Type and select **ShadingGroup** from the Foundry nodes.

You can jump inside the ShadingGroup in the same way as the NetworkMaterialCreate, by holding **Ctrl** and clicking with the **Middle-mouse button** or by clicking the enter node button.

You can either create nodes directly inside the ShadingGroup or you can cut-and-paste existing nodes from your network, into the group.

To connect shading nodes within the group to input and output sidebars:

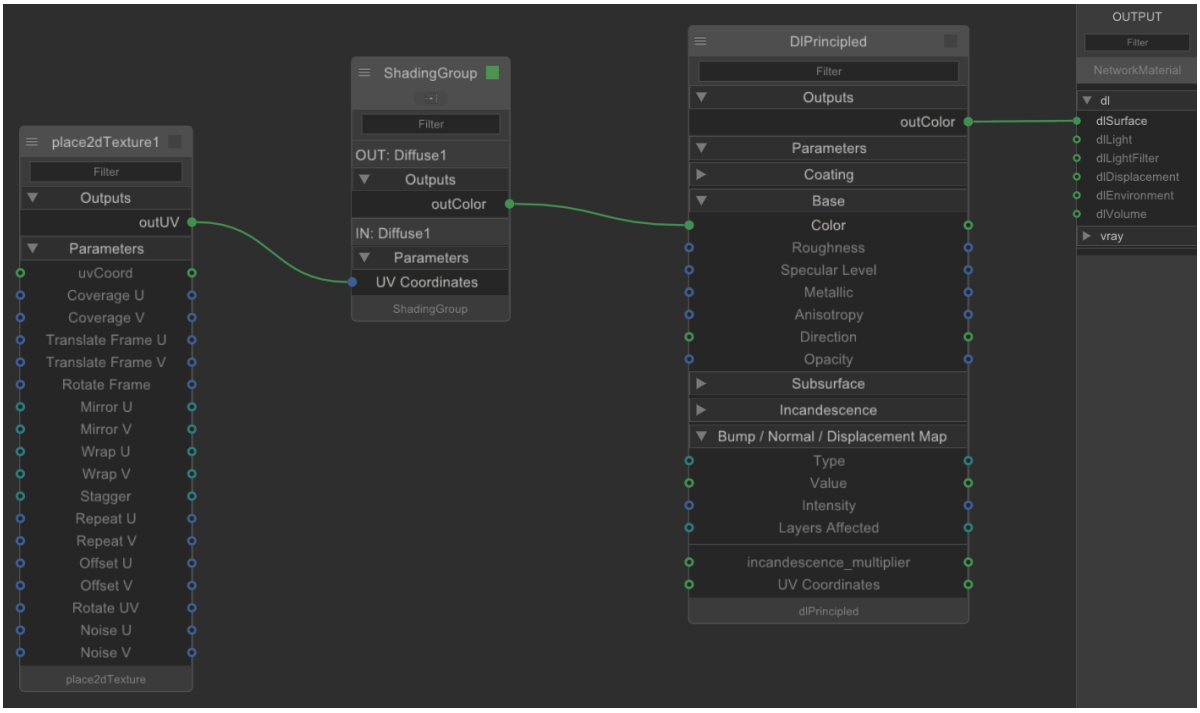
1. From your shading node, click the input parameter that you would like to be fed in.
2. Click anywhere on the fixed **INPUT** bar on the left to create a port and path for that specific parameter.
3. Click on the output parameter that you would like to be read out.
4. Click anywhere on the fixed **OUTPUT** bar on the right to create a port and path for that specific parameter.



Connecting a shading node within a ShadingGroup to the INPUT and OUTPUT sidebars

Tip: The **INPUT** and **OUTPUT** bars are filterable in the same way as the terminals on a NetworkMaterialCreate node. Enter a string in the **Filter** bar to search.

When you exit a ShadingGroup node, you'll see that the input and output parameters are visible on the outside of the node. You can now connect these up to the rest of your network.



Connecting the ShadingGroup node to the rest of the Network

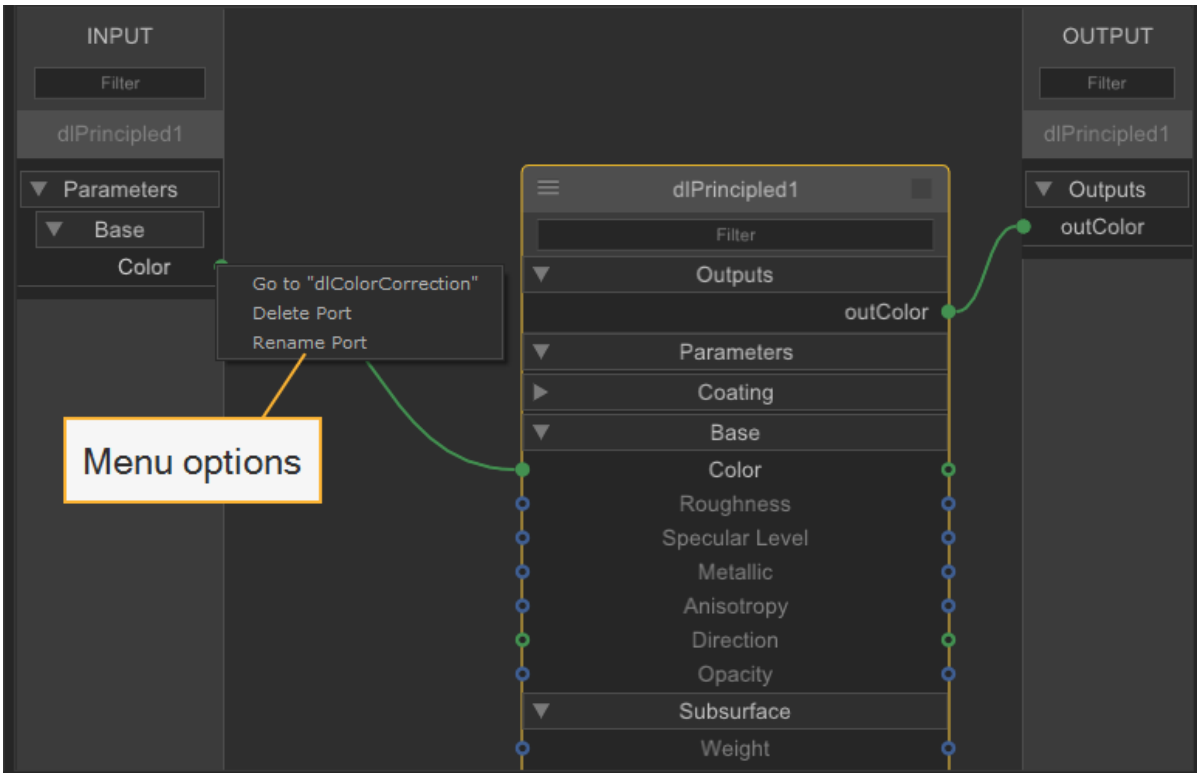
In the same way as any node within a NetworkMaterialCreate node, you can collapse and expand the ShadingGroup using **Alt + 1, Alt + 2** and **Alt + 3**. This helps to keep the whole section organized.

Tip: When collapsed, hovering over the shading node while pressing and holding the **X** key will temporarily expand it. It will collapse when released.

Menu options

From inside the ShadingGroup, right-click on one of your ports to bring up the options menu. From here, you can:

- Jump to the node it's connected to.
- Delete the port.
- Rename the port.



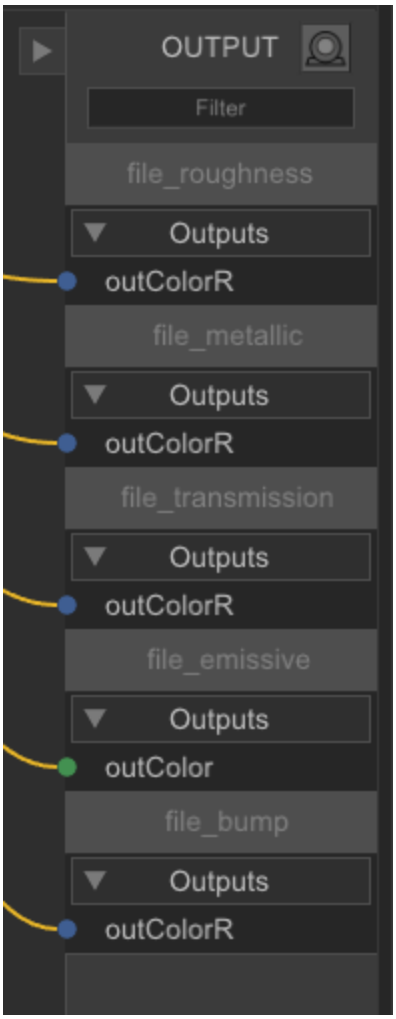
ShadingGroup port menu options

Tip: When renaming a port, you can use fullstops in between names to create nested pages. This is a great way of customizing your interface for input and output ports.

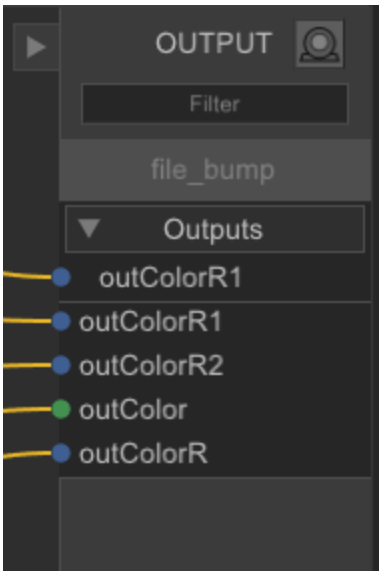
Rearranging Ports Within ShadingGroups

You can rearrange the order of your ports within a ShadingGroup, allowing you to customize its inputs and outputs. Rearranging ports allows you to organize your ShadingGroups easily. Changes made to the order of Input and Output ports made within the ShadingGroup are reflected on the ShadingGroup node when viewed within a Network Material context, such as NetworkMaterialCreate or NetworkMaterialEdit.

Rearrange Input and Output ports by middle-mouse clicking the port you wish to move and then dragging it within its respective sidebar. Nested or grouped ports can also move between or out of groups by middle-mouse dragging the chosen port between groups.

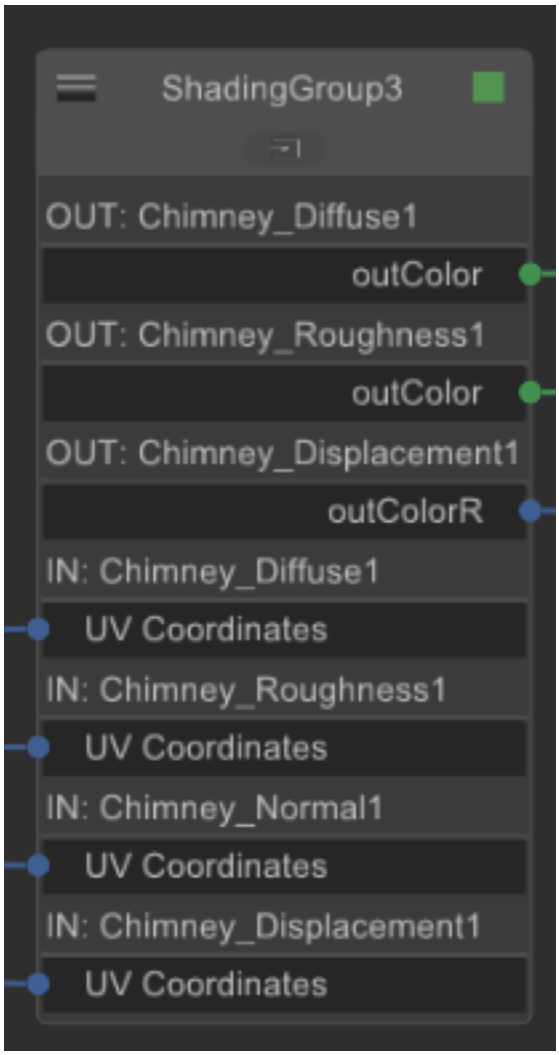


Ports on the **Output** sidebar before being rearranged.



Ports on the **Output** sidebar after being rearranged.

The order of rearranged ports is reflected in the ShadingGroup node once you exit the group.



ShadingGroup with default port arrangements.



The same ShadingGroup, with ports rearranged.

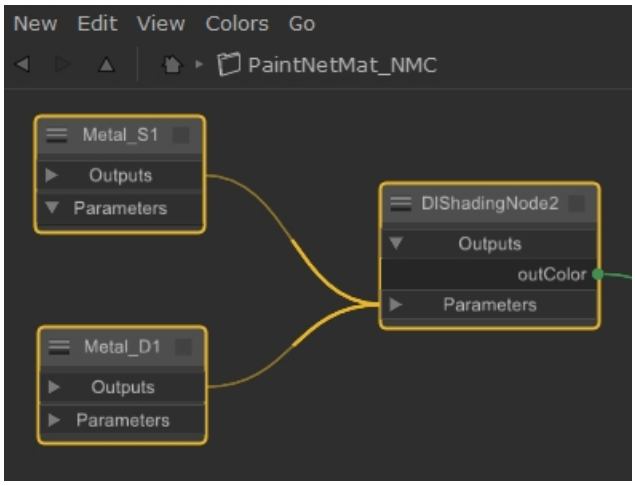
Note: You can rearrange ports within LiveShadingGroups. However, if the LiveShadingGroup is locked, ports cannot be rearranged or edited.

Sharing and Reusing ShadingGroups with Macros

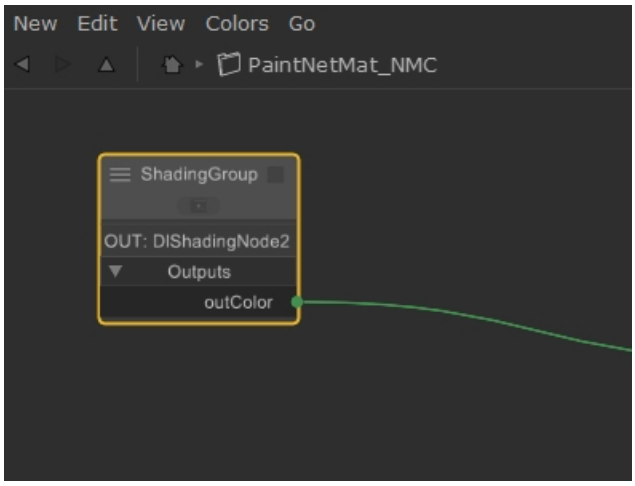
Macros enable you to wrap any single node, or ShadingGroup, and publish them so that their state is saved and they can be recalled in other shading networks. Saved macros can be added to a shader network as you would add a regular node, including from the **Tab** node creation menu.

To create a macro in a shading network:

1. Select the part of the shading network you want to save as a macro.
2. Press **G** on the keyboard to convert the selected nodes into a ShadingGroup.



The target shading nodes in the tree



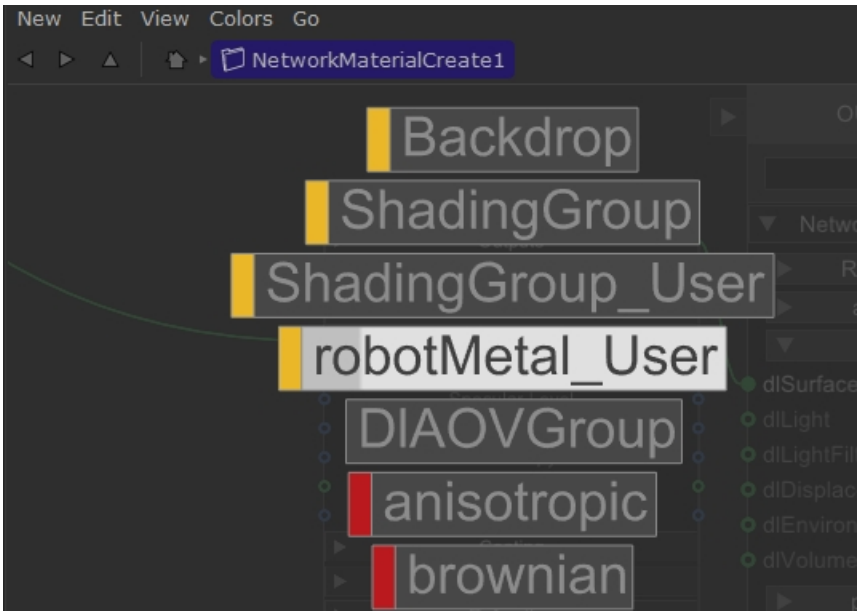
The target nodes collapsed into a group

Note: If you're creating a macro from a single node, you don't need to create a group.

3. Double-click the group, or use the keyboard shortcut **E**, to open its controls in the **Parameters** tab.
4. Click the wrench icon and select **Save as Macro**.

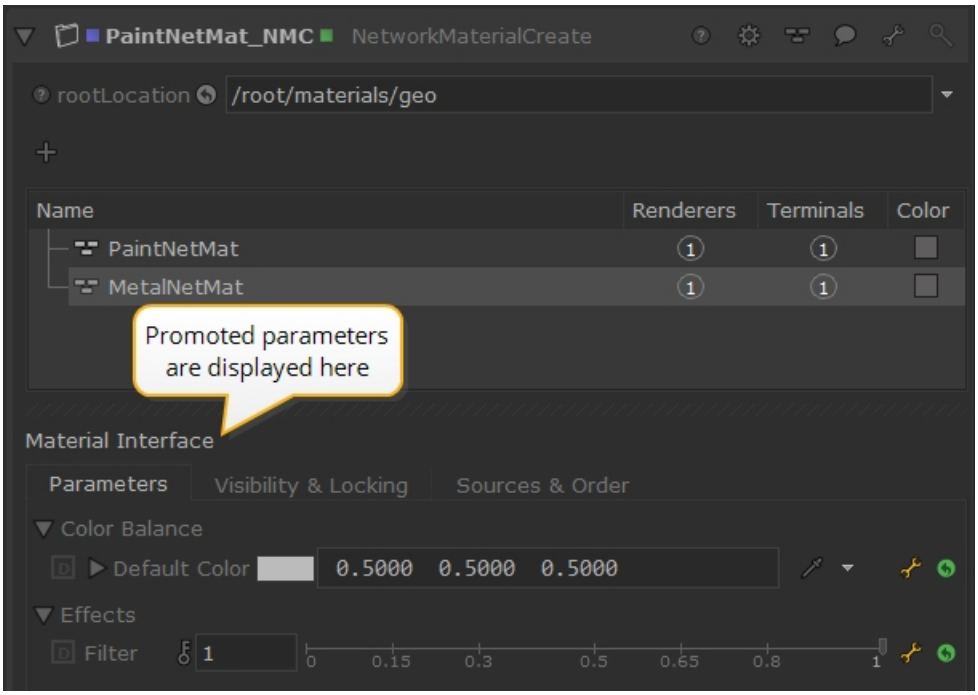
   By default, macros are saved in your home directory in **.katana/Macros/_User** and are automatically assigned the suffix **_User**. See Macros for more detailed information.

5. To add a macro to your shading network, press **Tab** on the keyboard and start typing the name of your macro.



Node Parameters and Interface Controls

The node parameters and interface controls are found in the **Material Interface** of the NetworkMaterialCreate and NetworkMaterialEdit nodes. These controls can be used to customize various shading node parameters that you may want to be accessible from outside Network Material nodes. The controls can be used to provide an artist-friendly set of parameters, established inside Network Material nodes, presented outside as an interface for controlling certain aspects without being exposed to all of the parameters. This can be beneficial as it allows the artist to focus only on the necessary parameters and make quick alterations without having to enter their Network Material nodes.



Promoted parameters in a NetworkMaterialCreate **Parameters** tab

Note: The examples on this page demonstrate promoting and customizing the material interface in the context of a NetworkMaterialCreate node. All these steps are reproducible when working with NetworkMaterialEdit.

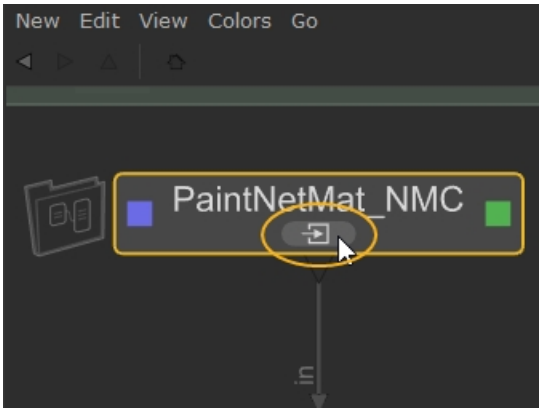
Adding Custom Parameters to the Material Interface

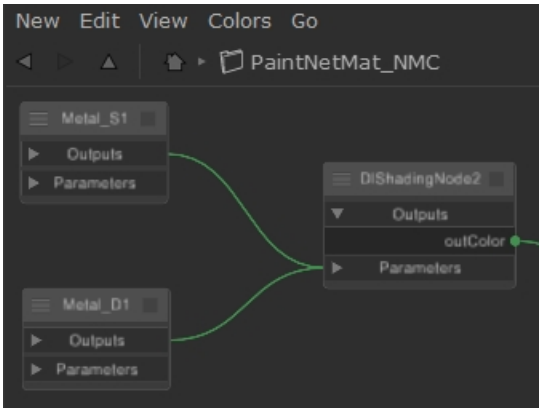
Network Materials can contain any number of nodes and opening the group to access node parameters is time consuming. If you know that some node parameters are used often, you can expose those internal node parameters on the parent NetworkMaterialCreate's **Parameters** tab for ease of access. You can promote parameters from any shading node within your NetworkMaterialCreate node, including those within ShadingGroups.

Note: The examples on this page demonstrate promoting and customizing the Material interface in the context of a NetworkMaterialCreate node. All these steps are reproducible when working with NetworkMaterialEdit, and there are no differences in workflows between the two nodes.

To promote parameters quickly with the default **Name**, **Page**, and **Label**:

1. Click the NetworkMaterialCreate node's expand button to display the nodes inside.





2. Double-click the node that contains the parameters you want to expose, or press the **E** keyboard shortcut.

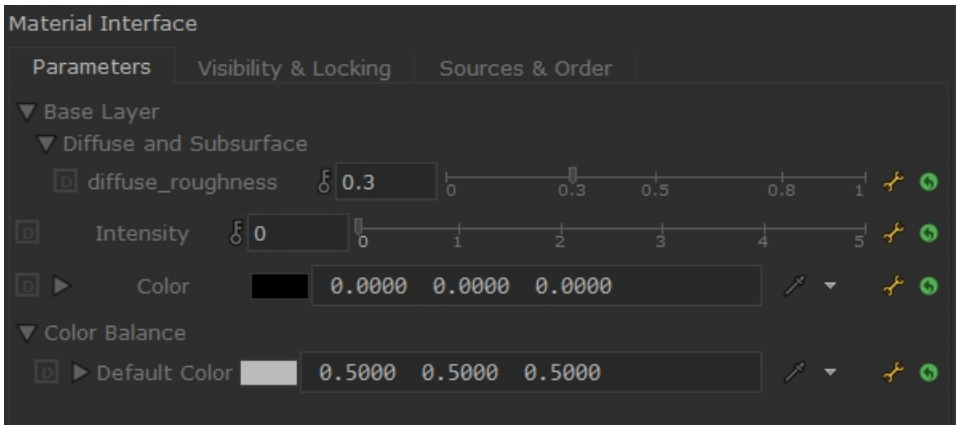
> **Tip:** If the source node is in a ShadingGroup, click the ShadingGroup node's expand button to display the nodes inside and then double-click the node that contains the parameters you want to expose..

   The node's **Parameters** tab opens.

3. Click the wrench icon to the right of the parameter you want to expose and select **Add to Material Interface**.

   The wrench turns yellow to indicate that the parameter is added to the **Material Interface**.

4. Open the **Parameters** for your NetworkMaterialCreate node to see your promoted parameters.



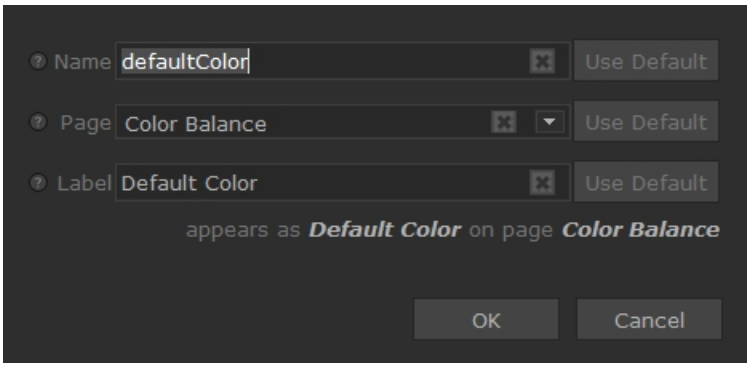
5. Click the wrench and select **Remove from Material Interface** to remove an exposed parameter.
6. Click the green arrow next to a parameter to open its source node's **Parameters**.

   Editing a parameter in the **Material Interface** or the source node updates both, so your nodes and NetworkMaterial are always synchronized.

To promote a parameter and customize its appearance:

1. Bring up the **Parameters** of your shading node by setting the edit flag.
2. Click on the **Edit Parameter** wrench icon on the right-hand side of the parameter you would like to promote.
3. Choose **Edit Material Interface Options** from the dropdown menu.

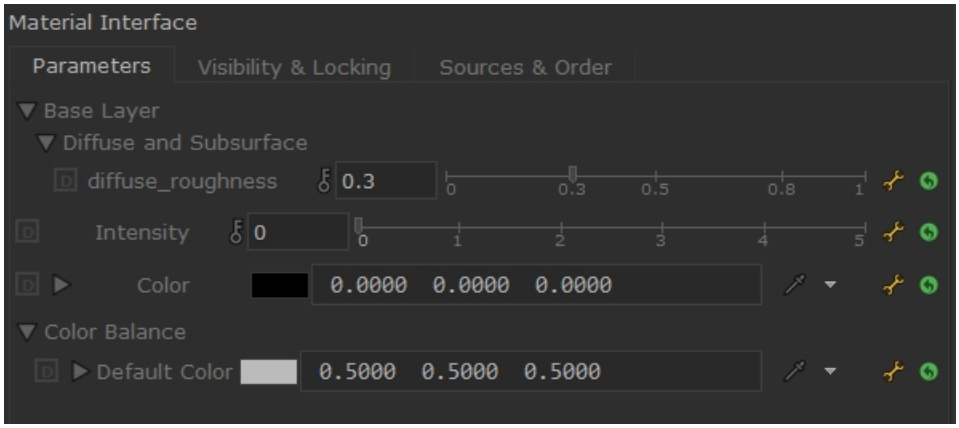
   The **Material Interface Options** dialog is displayed.



4. In the dialog box, enter the name you would like to give to the parameter in the **Name** field.
5. You can use the **Page** and **Label** fields to customize the appearance of the parameter.
   - **Page** - creates a subsection in the **Parameters** tab and places the new parameter in that subsection. If you leave **Page** blank, the new parameter is added directly to the **Parameters** tab.
   - **Label** - specifies the label describing the new parameter.

Tip: Pages appear as dropdowns in the **Parameters** tab. To create a page within a page, put a full stop between the two names. For example, **Base Layer.Diffuse and Subsurface** creates a page called **Base Layer** and a subsection called **Diffuse and Subsurface**.

6. Click **OK** to finish editing.
7. Open the **Parameters** for your NetworkMaterialCreate node to see your promoted parameters.



8. Click the wrench and select **Remove from Material Interface** to remove an exposed parameter.
9. Click the green arrow next to a parameter to open its source node's **Parameters**.

   Editing a parameter in the **Material Interface** or the source node updates both, so your nodes and NetworkMaterial are always synchronized.

Adding Prefixes to Custom Parameters from ShadingGroups

Promoting parameters from inside a ShadingGroup to the **Material Interface** allows you to add prefixes to parameters' **page** and **name**. This allows you to group parameters quickly without drilling down into the ShadingGroup to change the page and name of promoted parameters manually.



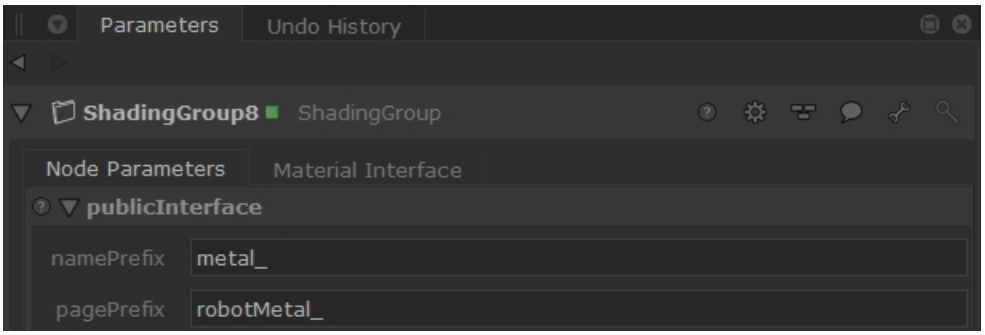
Note: The parameter **name** is its scripting reference, not its label in the **Material Interface**.

To add a prefix to promoted parameters:

1. Open the **Parameters** for your ShadingGroup node.
2. Click the **publicInterface** dropdown to open the prefix panel.
3. Enter the required prefixes in the available fields:

- **namePrefix** - adds a prefix to the promoted parameter's name. This is the parameter's scripting reference, not its label in the **Material Interface**.
- **pagePrefix** - adds a prefix to the page name visible in the **Material Interface**.

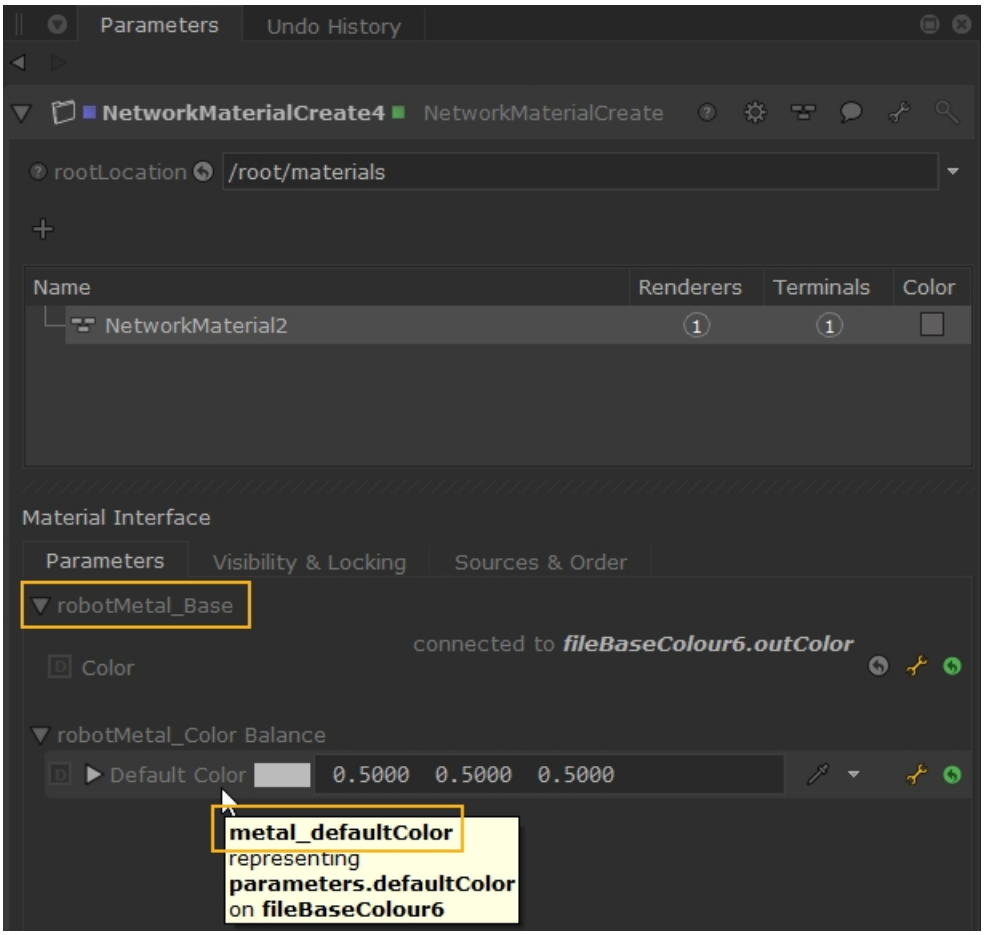
For example:



4. Open the NetworkMaterialCreate node's **Parameters**.

   The promoted parameters are prefixed as described.

> **Note:** The **name** prefix is only displayed when you hover over a parameter.

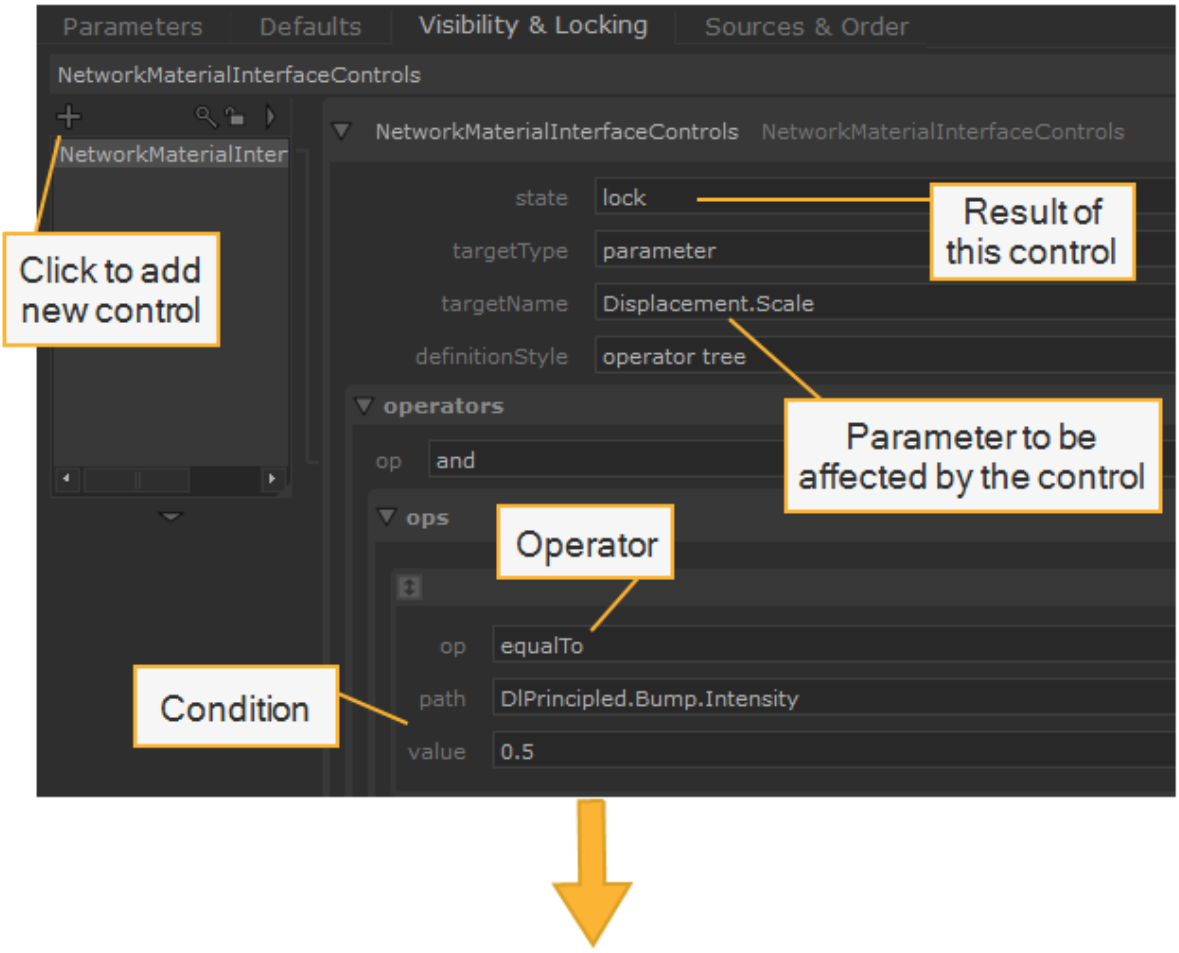


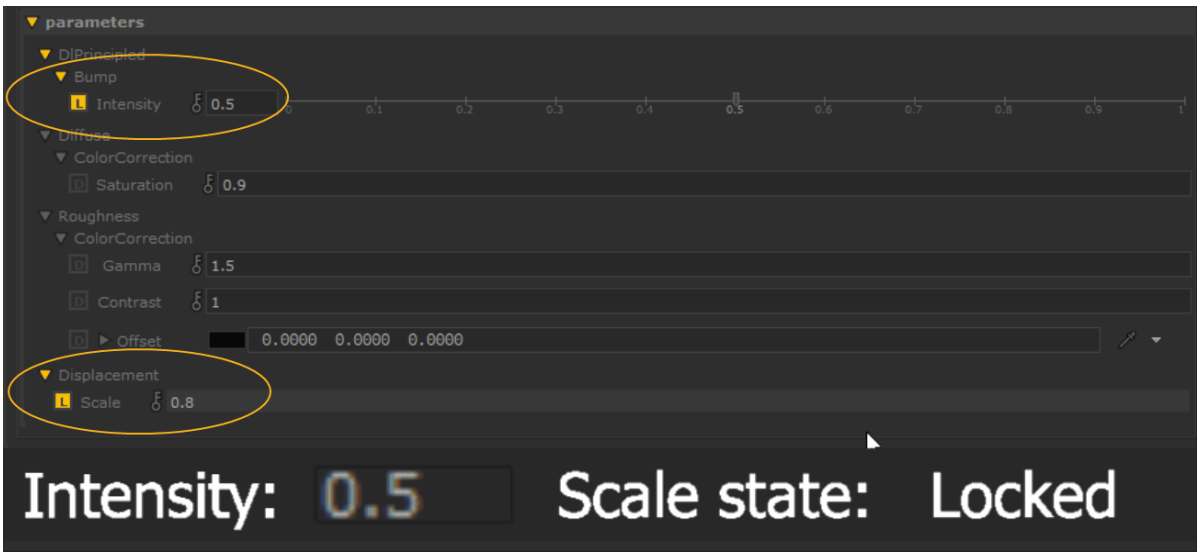
Adding Visibility and Lock Constraints to Parameters

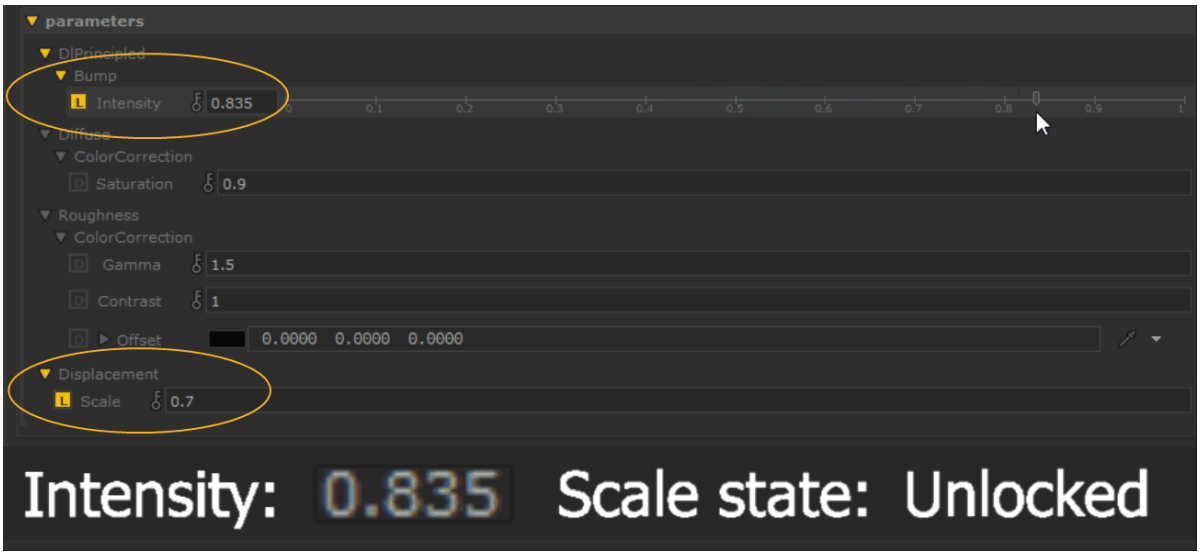
The **Visibility & Locking** tab uses a GroupStack node that automatically sets the type to NetworkMaterialInterfaceControls. It can manage any number of controls from within the parameters of a NetworkMaterialCreate node.

Visibility & Locking allows you to lock or hide promoted parameters so they cannot be changed under certain conditions. The conditions depend on the operator and the value of another promoted parameter. You can add multiple conditions and operators to customize the public parameter interface exactly as you need.

In this example, a control has been set up that causes the parameter **Displacement.Scale** to lock on the condition that the value of the parameter **Dlprincipled.Bump.Intensity** is equal to 0.5.


Parameters
Defaults
Visibility & Locking
Sources & Order
NetworkMaterialInterfaceControls
NetworkMaterialInter
NetworkMaterialInterfaceControls
NetworkMaterialInterfaceControls
state
lock
targetType
parameter
targetName
Displacement.Scale
definitionStyle
operator tree
operators
op
and
ops
op
equalTo
path
DlPrincipled.Bump.Intensity
value
0.5
Click to add new control
Result of this control
Parameter to be affected by the control
Operator
Condition


parameters
DlPrincipled
Bump
Intensity
0.5
Diffuse
ColorCorrection
Saturation
0.9
Roughness
ColorCorrection
Gamma
1.5
Contrast
1
Offset
0.0000 0.0000 0.0000
Displacement
Scale
0.8
Intensity: 0.5
Scale state: Locked



The result of a lock on the promoted parameters

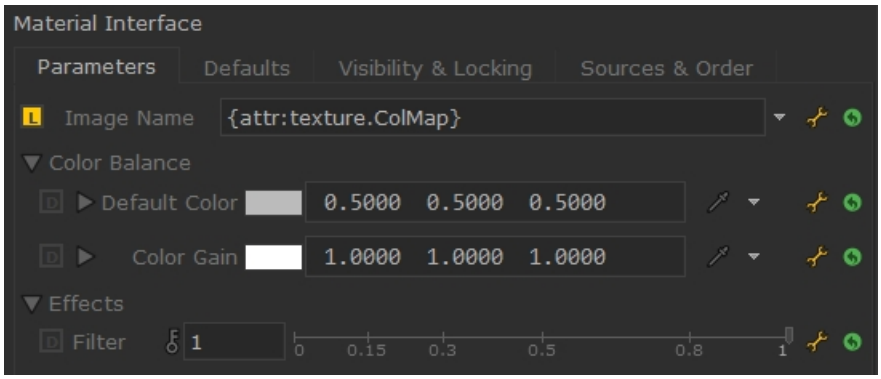
To create a new Network Material lock or visibility constraint:

1. Open the **Parameters** for your NetworkMaterialCreate node.
2. Click the **Visibility & Locking** tab.
3. Click the Add button to create a new control.

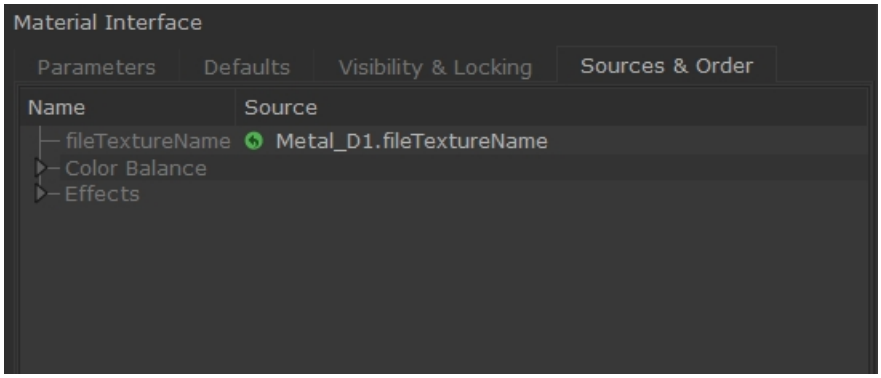
Note: For information about each parameter, see NetworkMaterialInterfaceControls.

Changing the Order of Promoted Parameters

Promoted parameters can be rearranged in whatever order you like. For example, you might want to position more commonly used parameters at the top of the **Parameters** tab or organize parameters alphabetically. Promoted parameters are listed in the order they appear on the **Sources & Order** tab.



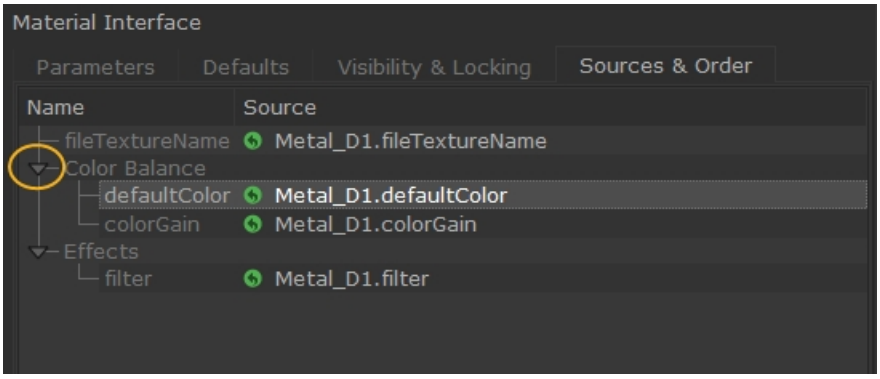
Default parameter order in the **Parameters** tab



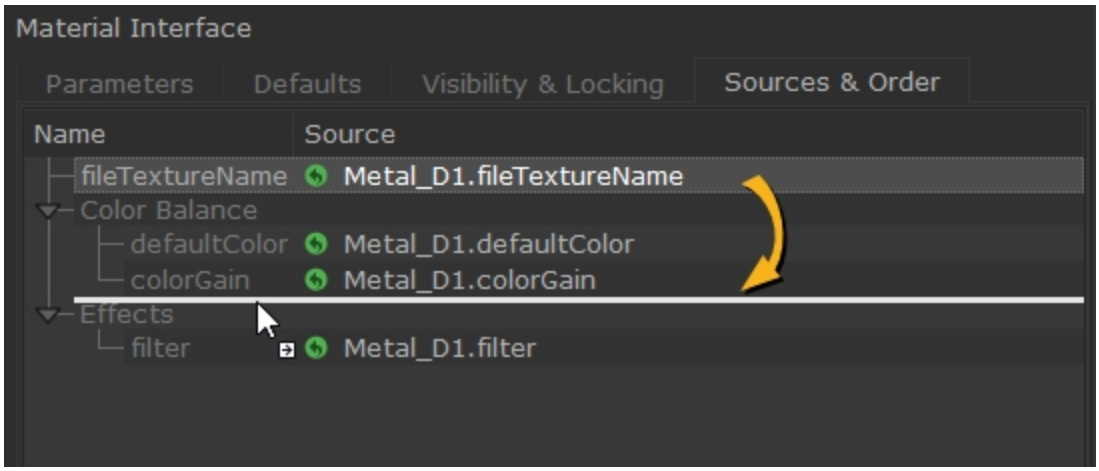
The same parameters in the **Sources & Order** tab

To change the order of parameters:

1. Click the **Sources & Order** tab in the **Material Interface**.
2. Click the dropdown to display the parameters hidden in pages, if required.

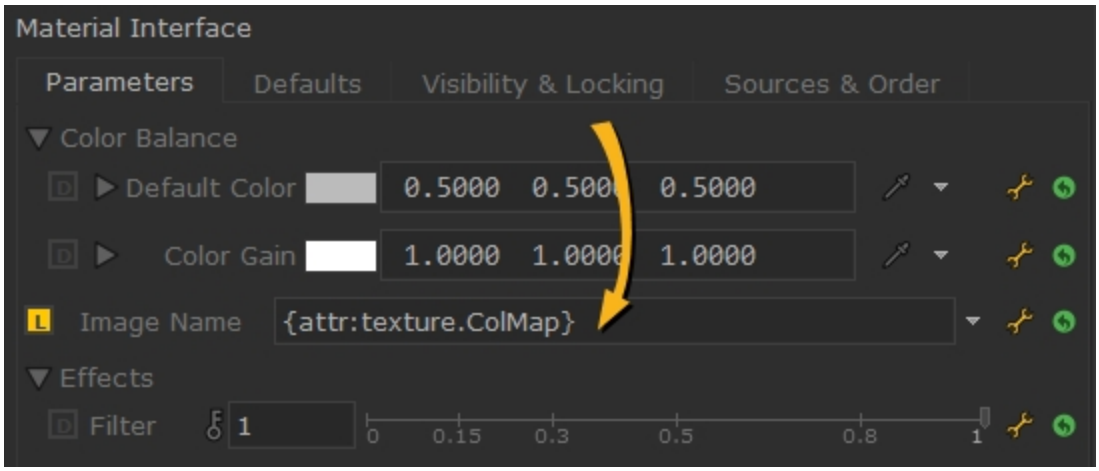


3. Middle-mouse click and drag to rearrange the order of the parameters. A white line highlights the parameter's new position as a guide.



Tip: You can drag and drop entire pages or single parameters, but parameters within pages can only be reordered within the page.

The **Parameters** tab updates to reflect the changes you made to ordering.

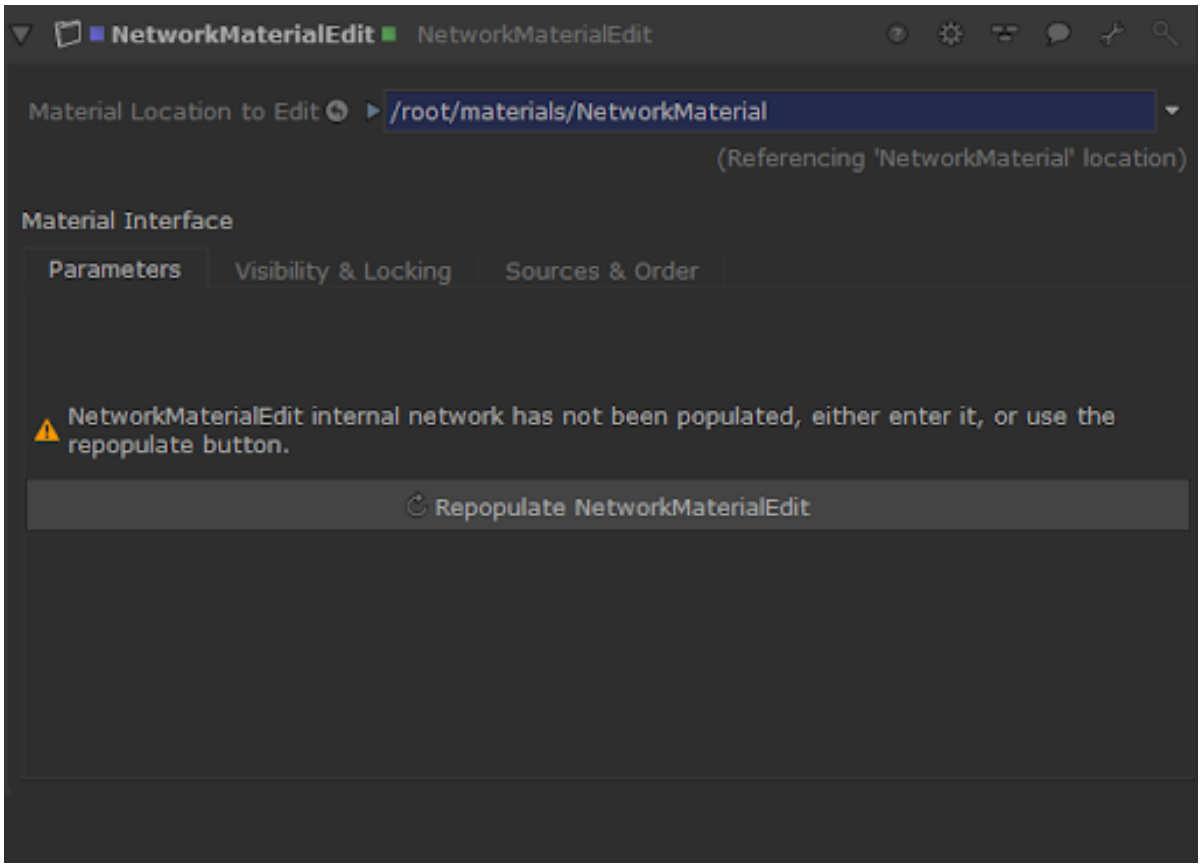


Using Custom User Interfaces With the NetworkMaterialEdit Node

When referencing an upstream material in a NetworkMaterialEdit for the first time, you must repopulate the NetworkMaterialEdit. Repopulating the NetworkMaterialEdit node generates any inherited shading networks and any promoted user parameters present in your upstream materials.

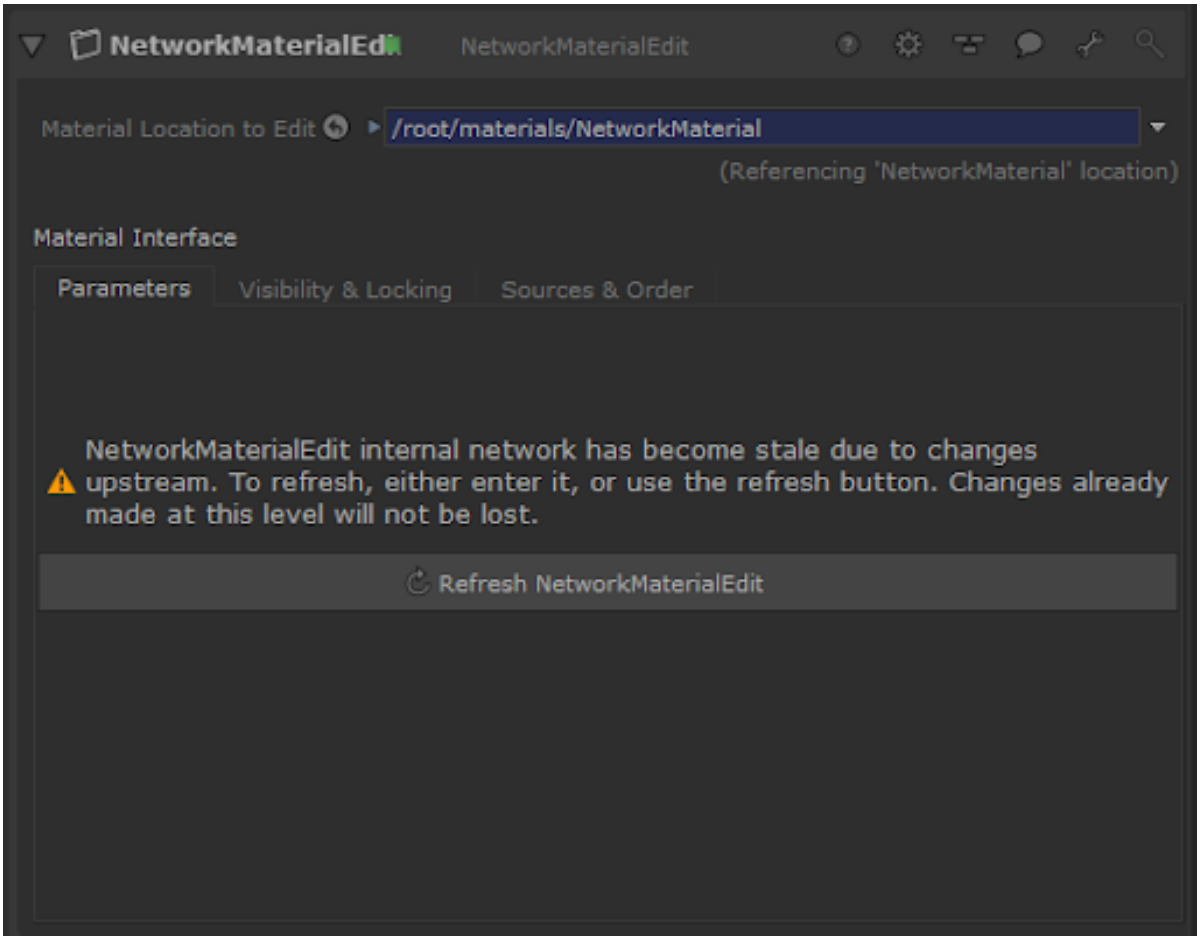
To repopulate a NetworkMaterialEdit node, either:

- Enable the NetworkMaterialEdit node's parameter flag from the nodegraph, or
- If the NetworkMaterialEdit node is already open in the **Parameters** tab, click the **Repopulate NetworkMaterialEdit** button.



Repopulating generates the shading network and the material interface is inherited from the material upstream.

If the contents of the upstream reference material are updated, you are prompted to refresh the contents of the NetworkMaterialEdit in the node's parameters.



Click **Refresh NetworkMaterialEdit** to include changes made in the referenced upstream material.

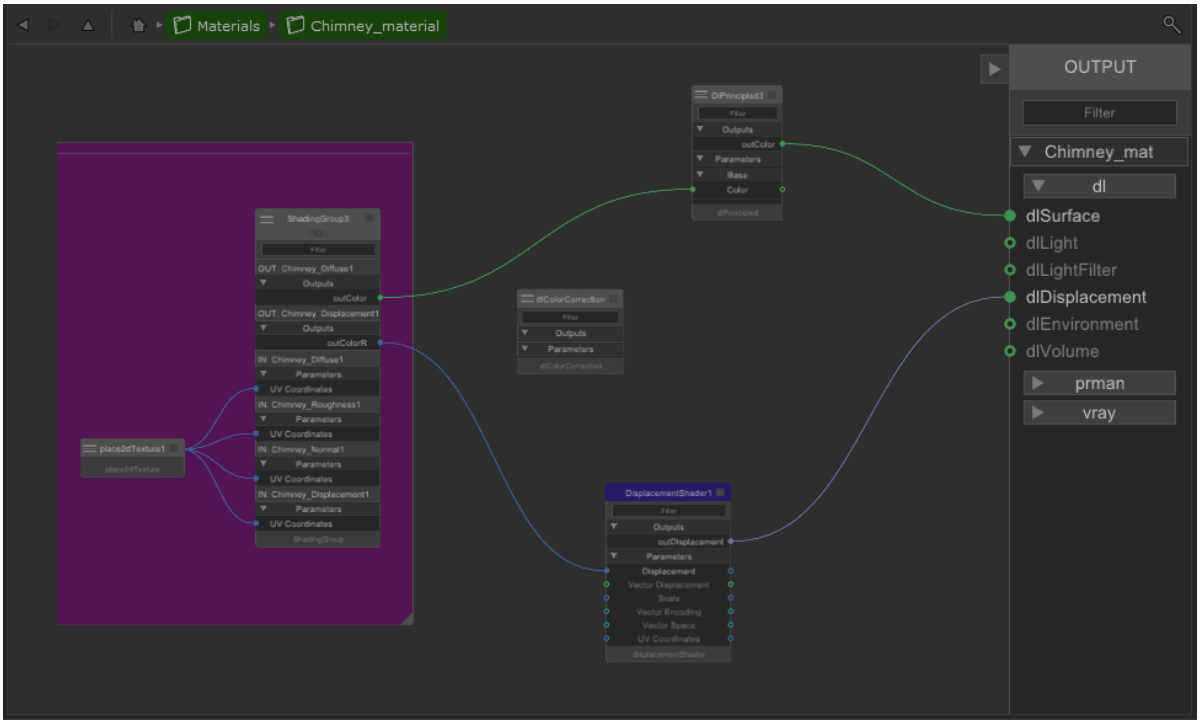
Tip: For more information on working with NetworkMaterialEdit, see Editing Materials With The NetworkMaterialEdit Node.

Editing Materials With The NetworkMaterialEdit Node

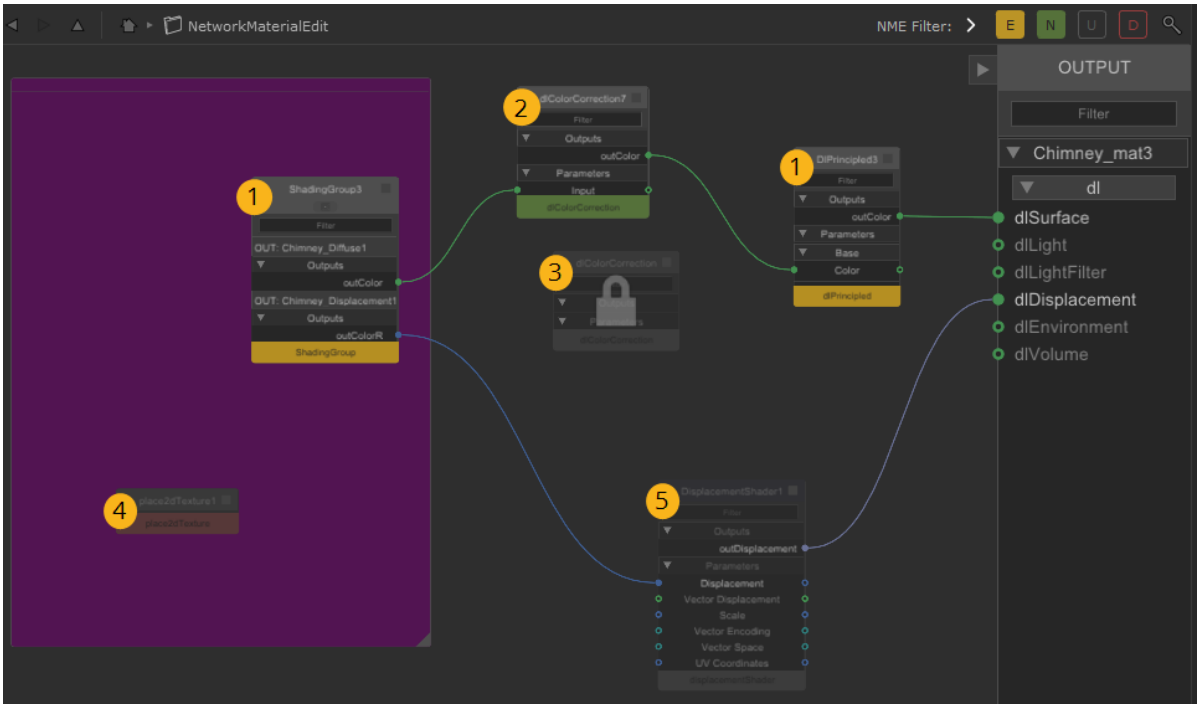
The NetworkMaterialEdit nodes allow artists to edit network materials, by adding or removing shading nodes from an existing network, as well as by creating new connections or breaking existing connections. You can also modify any of the parameters of shading nodes in an existing network.

The NetworkMaterialEdit node uses a **Scene Graph location**, set in the **Node Parameters**, which stores information about how a Network Material was created using the **material.layout** attribute. The **Scene Graph location** could be brought in through a **look file** and could have been created in a different department, with the original material network in a separate project. This workflow allows artists to have full control of the Network Material without needing to go back and edit the original material network.

Any upstream changes to the original material network change the **material.layout** attribute causing the NetworkMaterialEdit node to update to reflect those changes.



Inside a NetworkMaterialCreate node

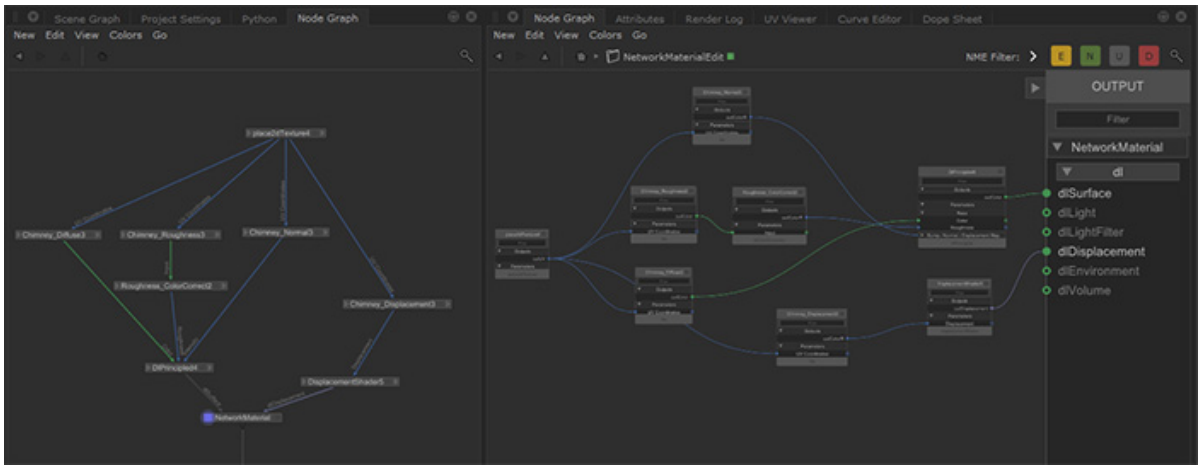


1. Edited nodes
2. New nodes
3. Locked nodes
4. Disconnected nodes
5. Unchanged nodes

Inside a NetworkMaterialEdit node

The contents of NetworkMaterialEdit nodes are drawn in the same style that was introduced for NetworkMaterialCreate nodes, with exposed ports on shading nodes and a left-to-right node layout. If the network material you are editing was created using a NetworkMaterialCreate node, the material network layout inside the NetworkMaterialEdit node is an exact duplicate of the original layout.

If the network material you are editing was created using the legacy NetworkMaterial workflow, the shading nodes in the NetworkMaterialEdit node are automatically arranged in the node graph.



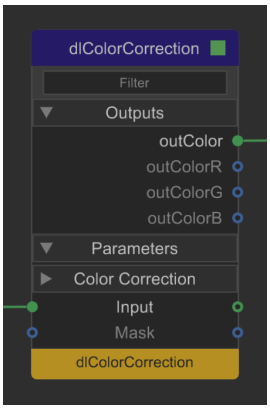
Using a NetworkMaterialEdit node to edit a legacy NetworkMaterial

Note: For more information on the NetworkMaterialCreate node, refer to Building Materials Using NetworkMaterialCreate.

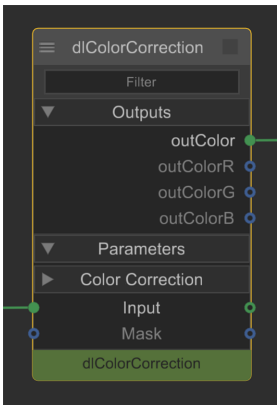
At first, the shading node network within a NetworkMaterialEdit node appears slightly dimmed to indicate that no changes have been made since the network was created, this can be changed using the NME Filter buttons.

Once a change is made to a node inside a NetworkMaterialEdit node, the node is marked with a yellow stripe to indicate that its parameters have been edited. Similarly, any new nodes are marked with a green stripe. This makes it easy to track any changes that you make within the NetworkMaterialEdit node, in comparison to the original NetworkMaterialCreate node.

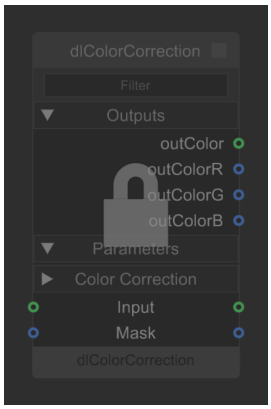
Nodes which are left disconnected from the output inside the original NetworkMaterialCreate node are disabled within a NetworkMaterialEdit node and marked with a padlock symbol to indicate they cannot be edited from with the NetworkMaterialEdit node. If you disconnect a node completely from within a NetworkMaterialEdit node, that node is marked with a red stripe.



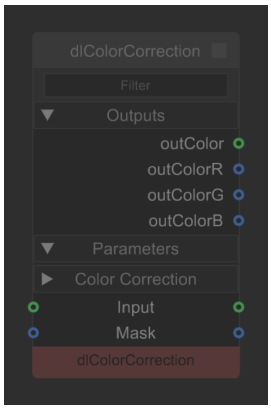
Yellow stripe indicates an edit.



Green stripe indicates a new node.



Padlock symbol indicates a disabled node.



Red stripe indicates a disconnected node.

This workflow allows artists to make adjustments to their shading node network without overriding the original. This can be useful when working on one asset across multiple shots as you are not constrained by how the original network material was set up, so small changes can be made on a per-shot basis.

NetworkMaterialEdit nodes combine the functionality of the existing NetworkMaterialParameterEdit and NetworkMaterialSplice node types, but in a UI that is visually representative of how the material was originally authored.

Filtering within NetworkMaterialEdit

The NetworkMaterialEdit node UI features the **NME Filter** options. These buttons allow you to dim nodes within the **Node Graph** for node states which are not toggled on. This is especially useful when working with large shading node networks as it allows you to focus your attention on a clear section of the node graph.



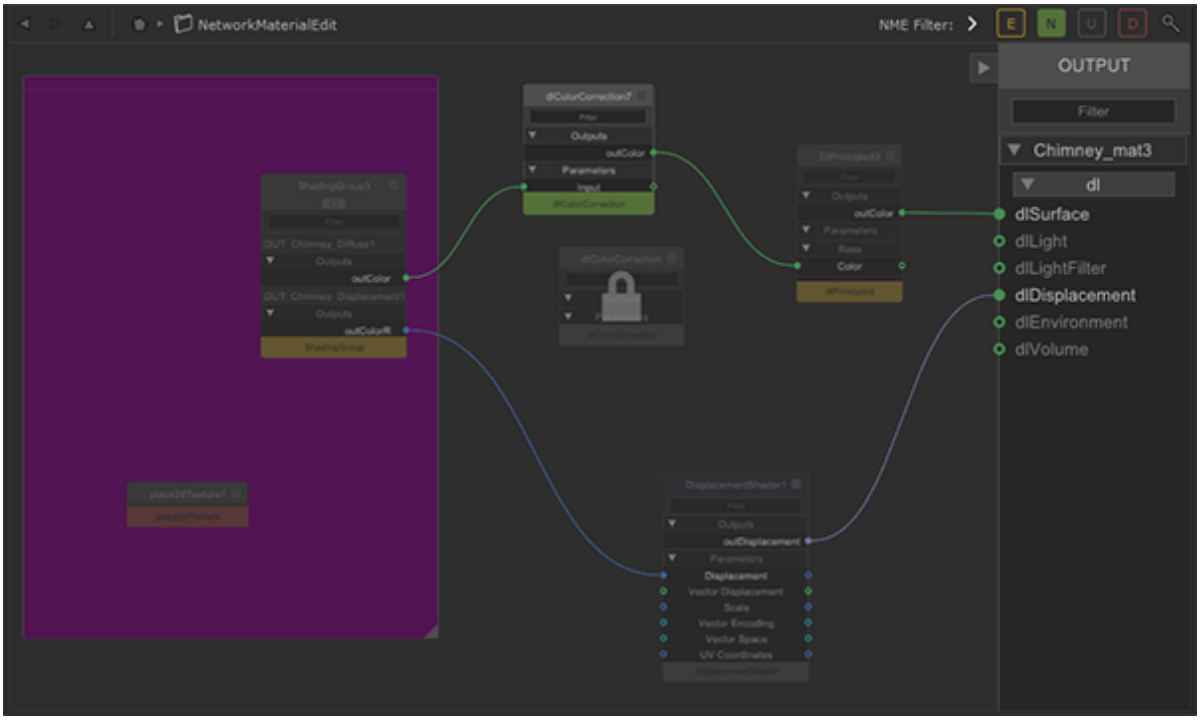
E - Edited nodes

N - New nodes

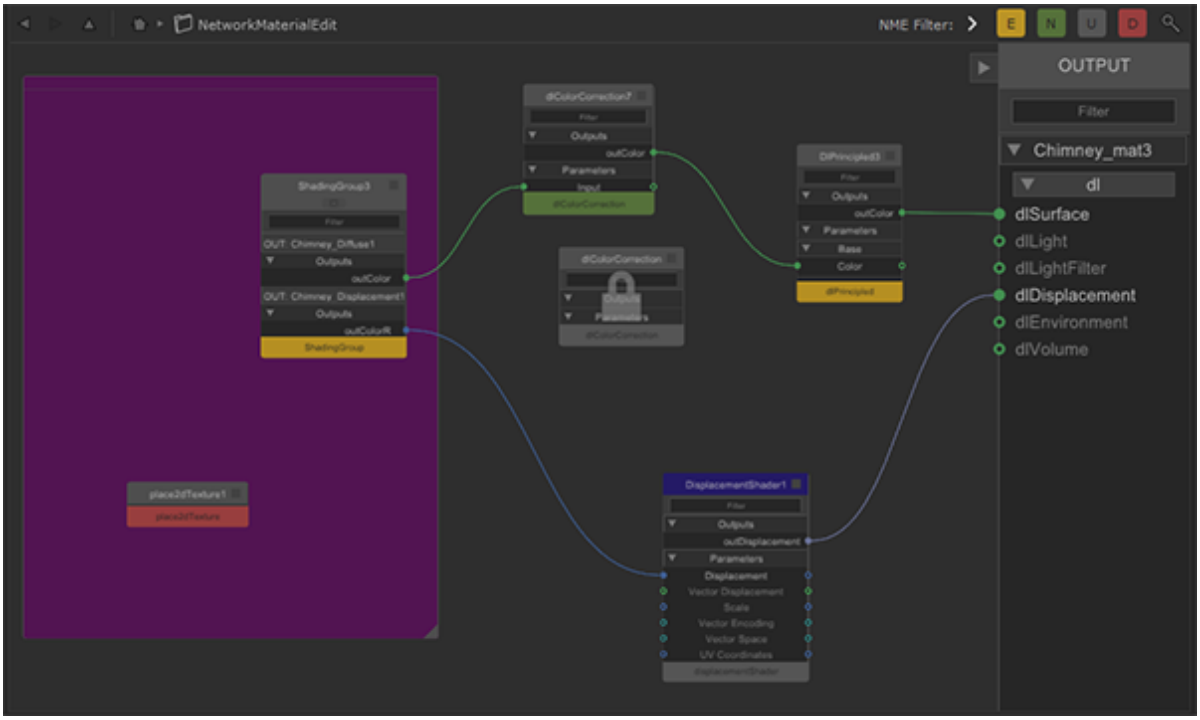
U - Unchanged nodes

D - Disconnected nodes

For example, to highlight which nodes are new to the NetworkMaterialEdit node, turn on the **New** button N and turn off the **Edited** E, **Unchanged** U, and **Disconnected** D buttons.



If you would like all nodes to be fully visible and none to appear dimmed, turn on the **Edited** E, **New** N, **Unchanged** U, and **Disconnected** D buttons.

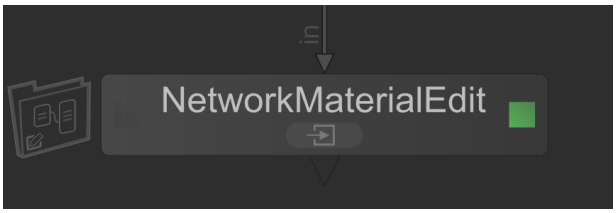


How to Use the NetworkMaterialEdit Node

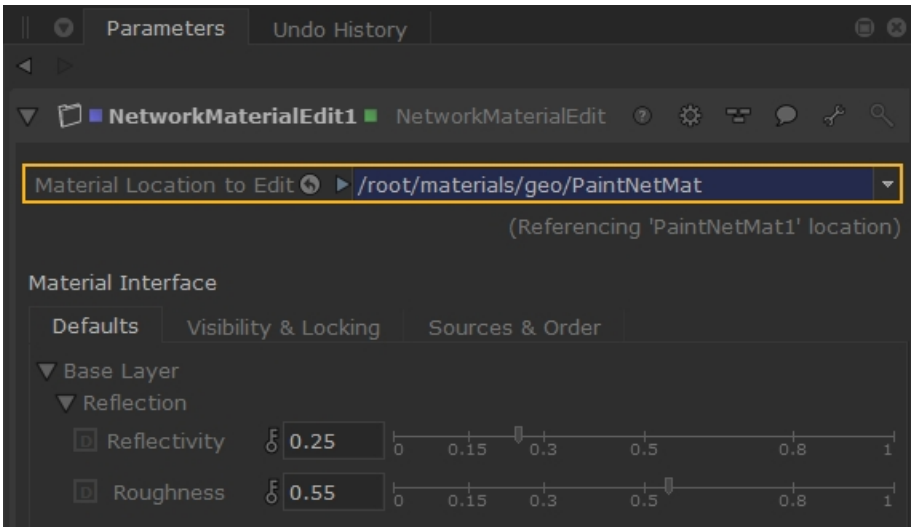
1. Create a NetworkMaterialEdit node by pressing **Tab** and typing **NetworkMaterialEdit**.
2. Connect the NetworkMaterialEdit node to your network.

> **Note:** The NetworkMaterialEdit node must be downstream of the original NetworkMaterialCreate node information, whether that information has come from a look file or the NetworkMaterialCreate node itself.

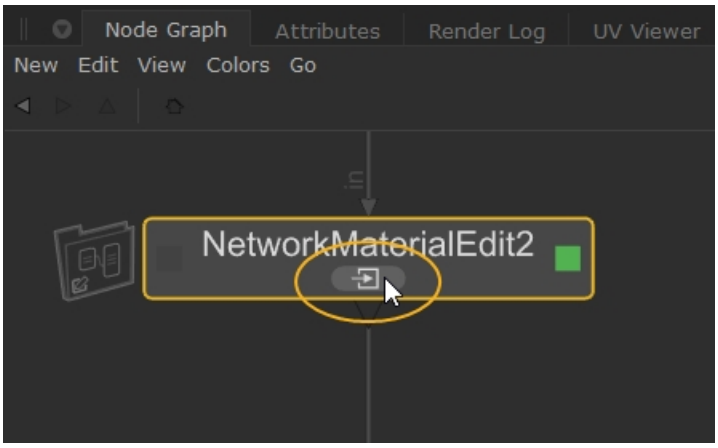
3. Open the **Parameters** of the NetworkMaterialEdit node by hovering your mouse over the node and pressing **E**, or by clicking the Edit flag on the node.



4. Use **Ctrl** + **Middle Mouse button** to drag the NetworkMaterial from the **Scene Graph** to the **sceneGraphLocation** field in the NetworkMaterialEdit **Node Parameters**.



5. Click the **Enter** button on the NetworkMaterialEdit node to view the shading node network and start editing.



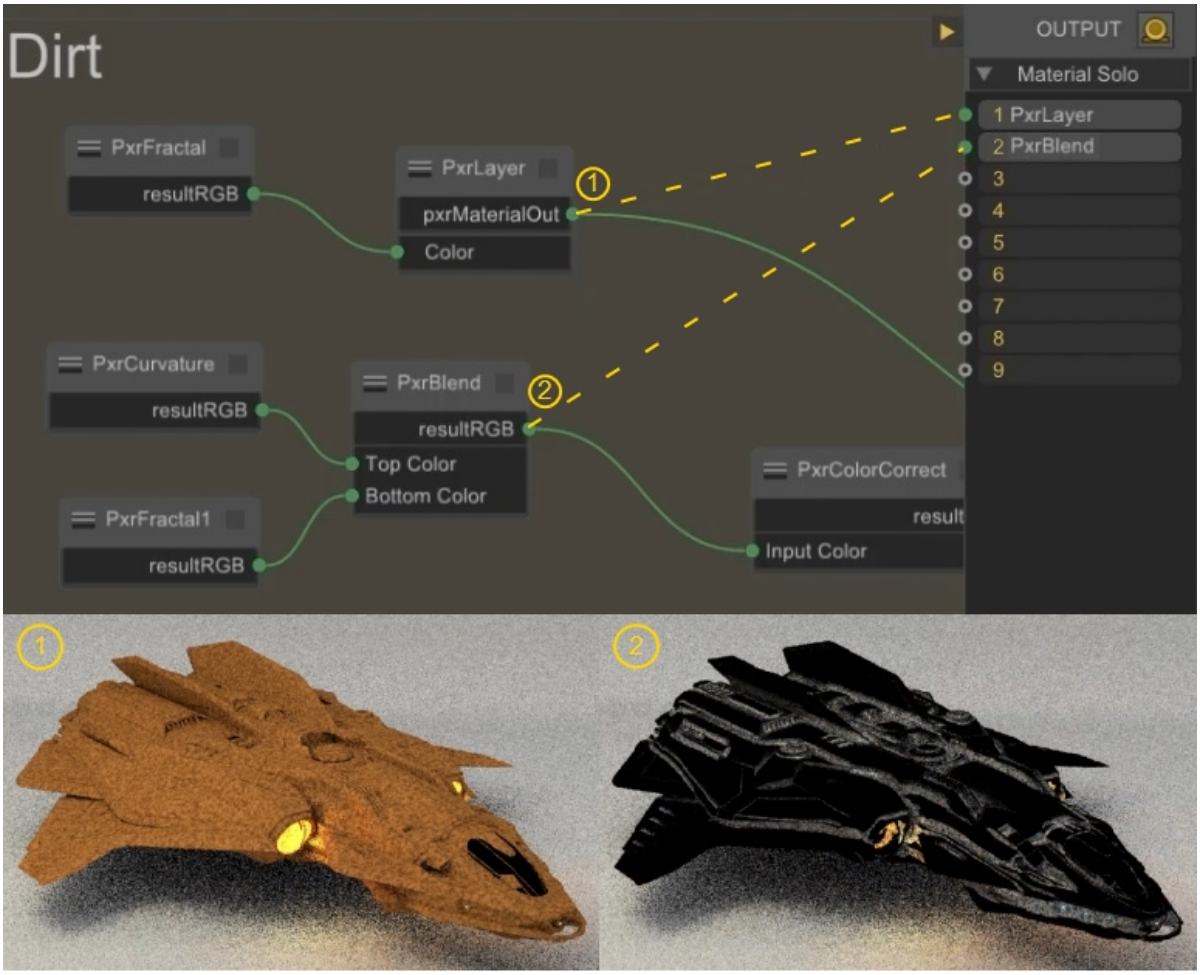
Note: Shading nodes within a NetworkMaterialCreate node that are not contributing to the final material are missing important attributes, making it hard to reconnect them from a NetworkMaterialEdit node. These nodes are locked from within the NetworkMaterialEdit node.

Easily Preview Sections of Your LookDev Using Material Solo

Material Soloing accelerates your look development by short-circuiting a shading network to preview sections of your materials independently. With an interactive Preview or Live render running, use the 1-9 keyboard shortcuts to preview different points of your network and the render updates to display the material output from that point in the node graph.

Note: Soloing only affects interactive, preview, and live render sessions. Disk and batch renders ignore soloing and render the entire scene.

You can map multiple nodes to different hotkeys, even those within ShadingGroups, so it's really quick and easy to switch between different points in a shading network to compare outputs. Soloing also helps you to familiarize yourself with networks handed to you from other artists or debug networks by locating any potential issues much faster.



Note: You can modify soloing behavior using Python. For example, you might not want to connect to the first output available for certain nodes. There are example MaterialSoloingConverters scripts for several renders shipped with Katana here: <install_dir>plugins/Src/Resources/Core/Plugins/

Enable Material Soloing

Soloing is activated from inside material networks and temporarily replaces the **Output** sidebar. To enable soloing:

1. Click the soloing icon in the **Output** sidebar, or press **0** (zero) on your keyboard.

   The **Output** sidebar changes to show the available soloing ports. Connecting a port displays the name of the node to which the port is connected.

2. To solo a material, drag the output from the target node into a port in the **Output** sidebar or drag a port from the **Output** sidebar to the target node,

   OR

   Select or hover over the target node and use keyboard shortcuts **1** through **9** to assign the node to the corresponding port.

   > **Tip:** Pressing the **1** through **9** keyboard shortcuts with soloing disabled activates soloing and connects the port automatically.

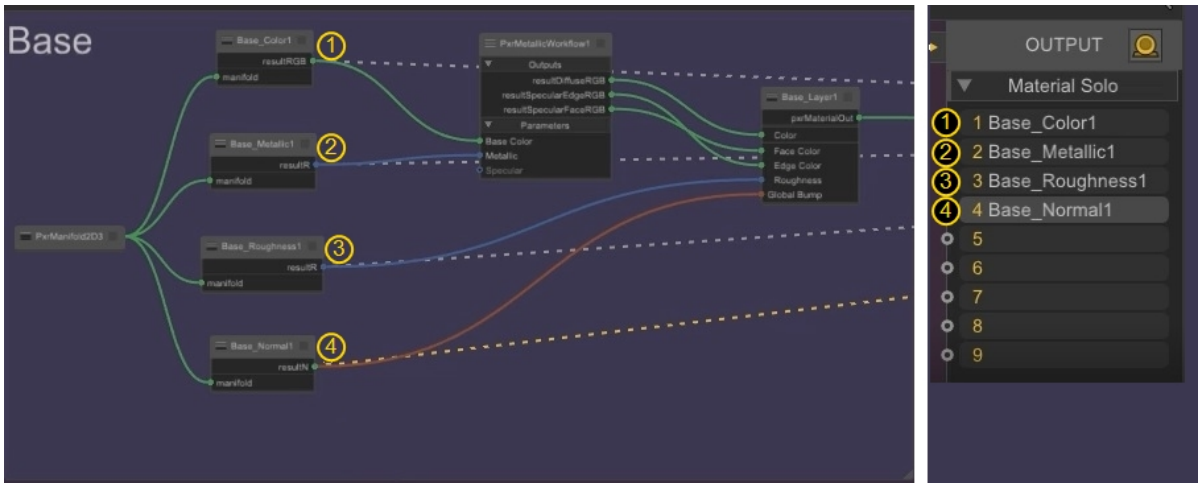
3. You can disconnect a port by double-clicking the required number in the **Output** sidebar or by disconnecting the pipe manually.
4. Click the soloing icon in the **Output** sidebar, or press **0** (zero) again to disable soloing mode and return to the **Network Material** view.

Familiarize Yourself with Another Artist's Work

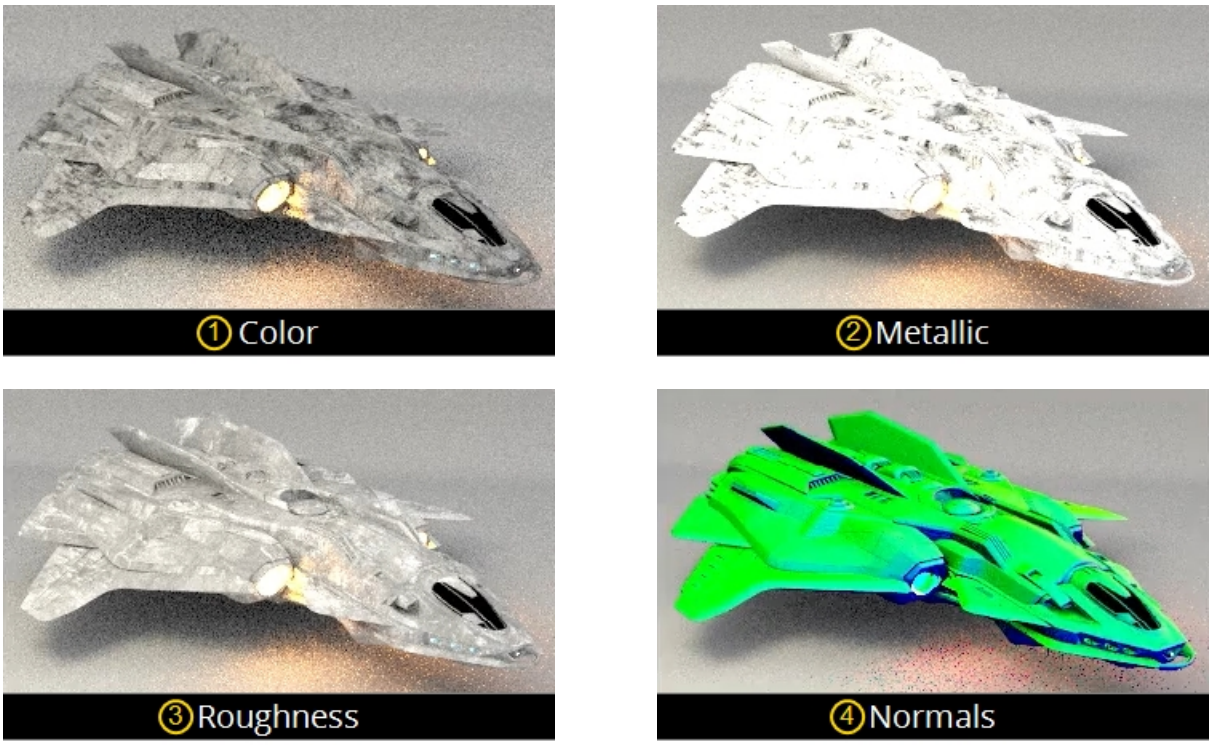
Collaboration is a key component of any pipeline, so working with materials that someone else created is very common. The first thing you'd do as an artist is make sure you understand how this material was set up and how each section of the node graph impacts the rendered result.

In this example, we'll preview some texture maps for a spaceship asset. Once you understand the setup, you can then develop the look of the asset consistently to retain the overall look of the shot.

1. Enable the material soloing tools by clicking the soloing icon in the **Output** sidebar, or by pressing **0** (zero) on your keyboard.
2. Connect the Color, Metallic, Roughness, and Normal outputs to soloing ports by hovering over the nodes and pressing the **1** through **4** keyboard shortcuts.



3. Cycling between the four ports gives you a clear picture of how each node contributes to the material.



4. You can also connect a port to the last node in the this section of the node graph to quickly view all the component parts merged together.

Tip: Switching to an unassigned port also displays the final material. In this example, ports 5 through 9 have no assignment so the final material is displayed instead.

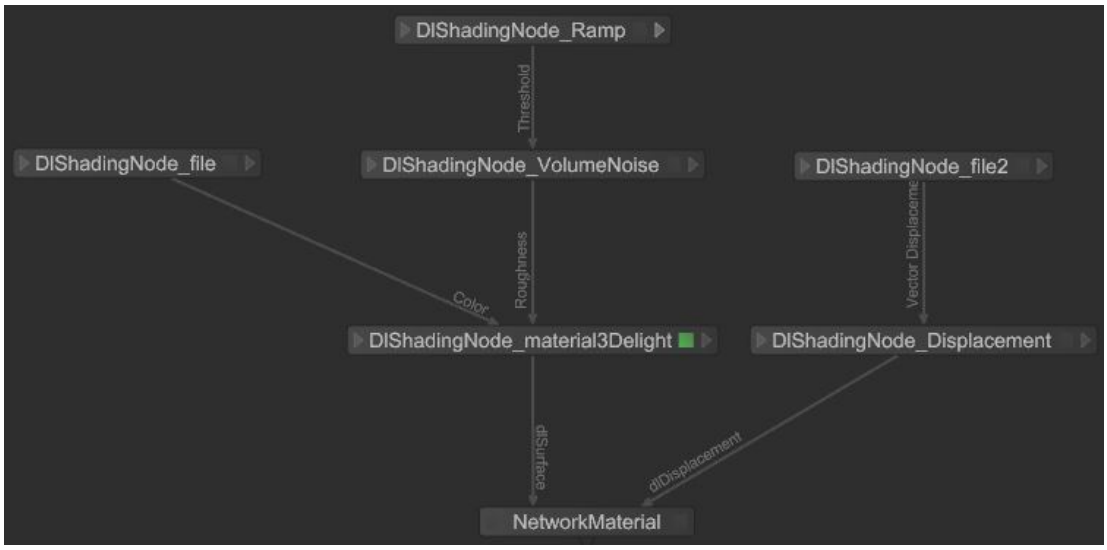
For example, you might want to have the final result in port 1 and the other materials in ports 2 through 5, so that you can always press 1 on you keyboard to see the result.

Working your way through a material network this way before you start work gives you a good understanding of the whole picture and helps to avoid wasting time figuring out what each component is doing to reach a final result faster.

Network Materials

Note: This is a legacy workflow, to learn about the new and improved NetworkMaterialCreate workflow, see Building Materials Using NetworkMaterialCreate.

Building a shader from a number of smaller parts is versatile and often more efficient. Complicated shading networks can be built from simple re-usable 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 DIShadingNode 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:

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 it's not strictly needed, as Katana handles name clashes gracefully, it's 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, so you don't need to add this in the **nameSpace** field. 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 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. These ports are shader-specific and multiple ports can be added to the same NetworkMaterial node. To add ports, click **Add Terminal** and select a port type from the terminal type dropdown.

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 DlShadingNode 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.

For example, when creating a 3Delight surface shader as a network material, the shader node that connects to the **NetworkMaterial** node's **dlSurface** 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

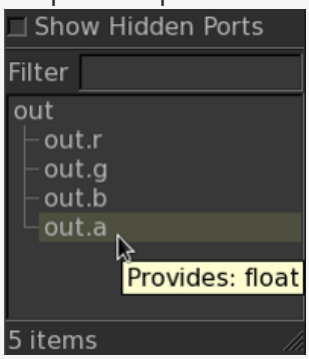
Connections between shading nodes support connection checking logic, so when connecting shading nodes together through the UI, only permissible connections are allowed. For example:

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**.

   For this example, 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.



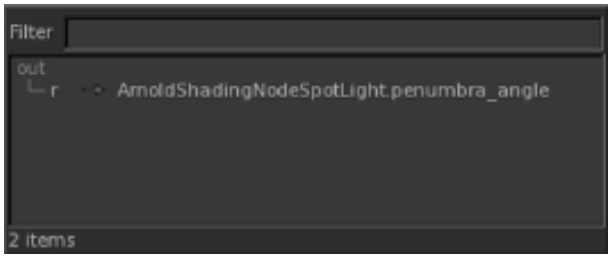
In the image above, you can see the **r** channel output of the image nodes provides a float.

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

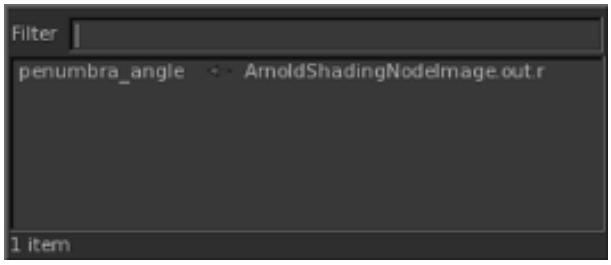
Note: To connect an output to an input, click on the output arrow of the source shading node, and click on the output you want. Then, click on the input arrow of the target shading node, and select the input you want to connect to.

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 to the image node, the following is displayed:



This indicates 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 to the **spot_light** shading node produces the following:



This indicates 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 the 3Delight renderer, the shading node is called dlShadingNode, and for the Arnold renderer, 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** dropdown below **nodeType**.

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

Connecting a Shading Node

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

1. Click the output arrow ▶ on the right side of the shading node.

   A list of possible inputs displays. Again, this list depends on the shader.
2. Select the output parameter from the list.

   This creates the output connection, highlighted in yellow, that follows your cursor until you click.
3. Click the input arrow ▶ on the left side of the next shading node.

   A list of possible inputs is displayed. The contents of the list depends on the shader.
4. Select an input from the list.

   This creates the input connection.

Using this method, it's 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.

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.

Connection Logic Checking

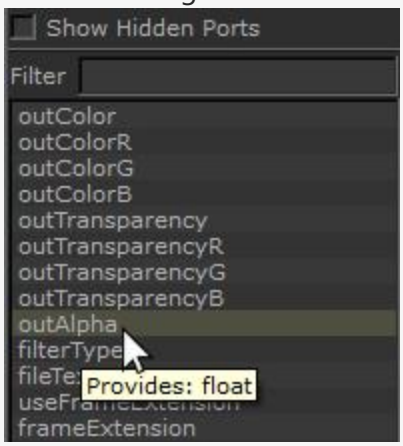
Connections between shading nodes support connection checking logic, so when connecting shading nodes together through the UI, only permissible connections are allowed. For example:

1. In an empty recipe, create two DlShadingNodes. Set one to nodeType **file**, and the other to nodeType **spotLight**.
2. In the **file** type shading node, set the **image name** parameter to point to an image.
3. Click on the right output arrow ▶ of the **file** shading node to see the available outputs.

   For this example, click on the **outAlpha** to select the alpha channel of the image shading node, then click on the left input arrow ▶ of the **spotLight** shading node.
4. A new window shows the connection options.

   The input ports on the standard node that cannot accept an **outAlpha** input (such as **Decay Rate**) are grayed out.

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



In the image above, you can see the **outAlpha** channel output of the image nodes provides **Float**.

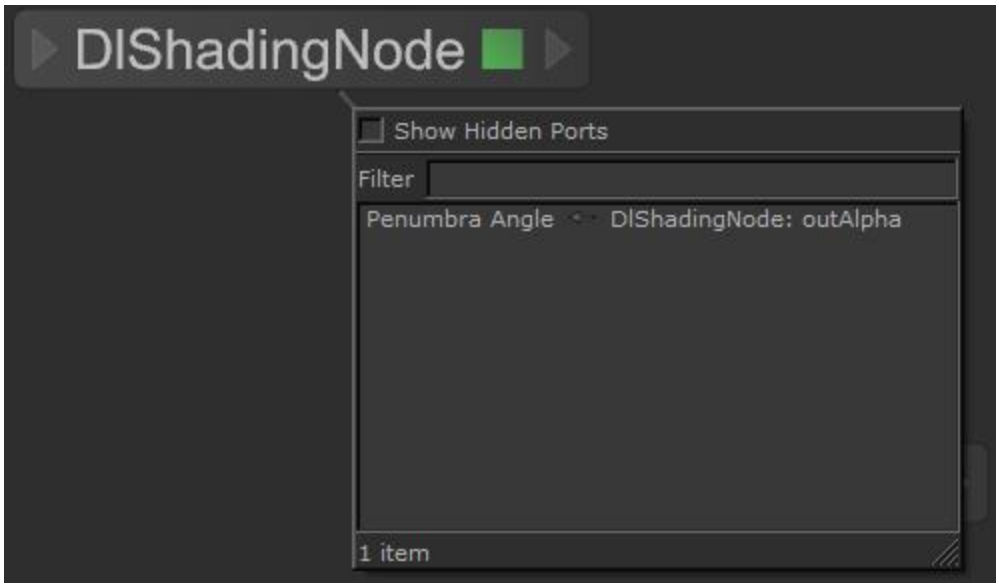
5. Connect the **outAlpha** channel output of the image shading node to the **PenumbraAngle** input of the **spotLight** shading node, which accepts float inputs.

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 to which you want to connect.

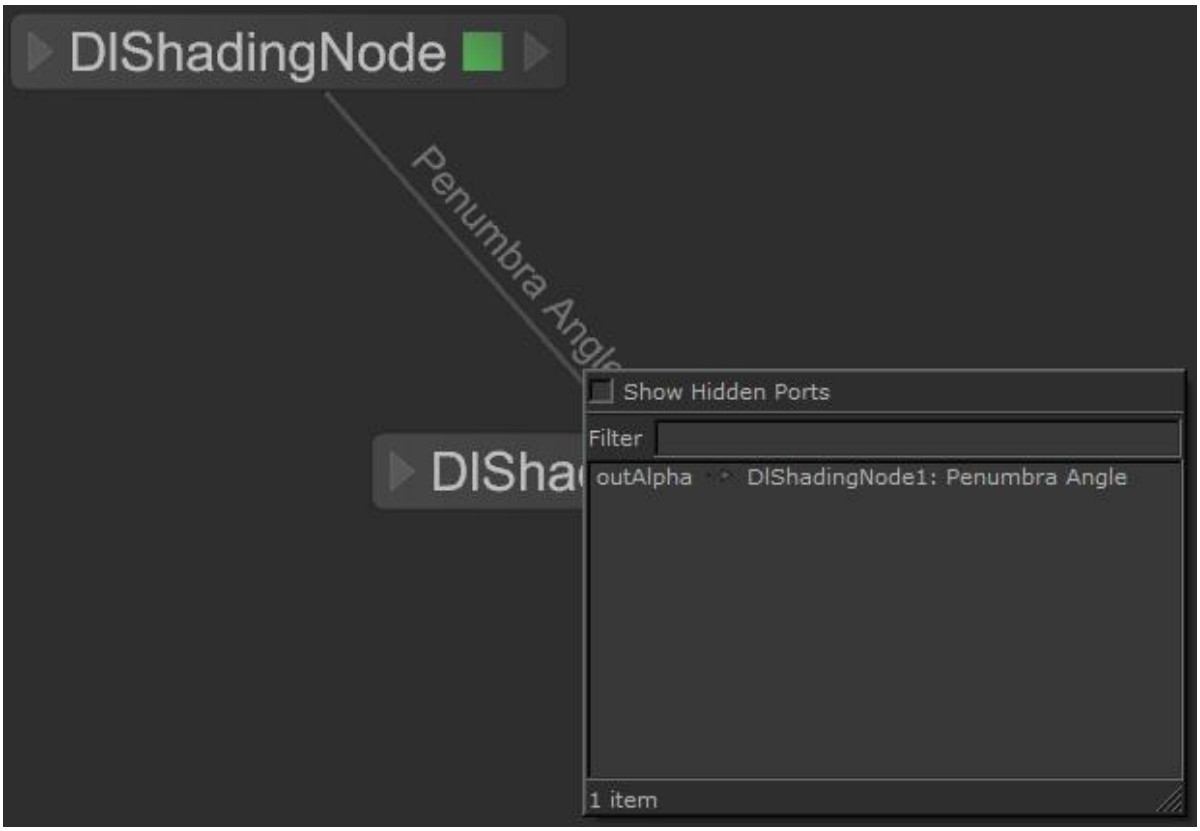
Showing Connections

Once two shading nodes are connected, they're joined in the node graph by an arrow. Left-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 and left-click on the arrow connecting the **image** and **spotLight** DlShadingNodes. When the mouse is closer to the image node, the following is displayed:



This indicates that the **outAlpha** output is connected to the **Penumbru Angle** input of a node named, in this case DlShadingNode1. Left-clicking with the mouse closer to the **spotLight** shading node produces the following:



This indicates that the **Penumbra Angle** input has an incoming connection from the **outAlpha** output of the (in this case) DlShadingNode.

Disconnecting a Shading Node

To disconnect one shading node from another:

1. Hover the mouse over the input connection and 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** tab.

Exposing a Shading Node's Parameters

Materials can be built from a large number of shading nodes, which might make it difficult to work out which parameters are important. Instead of trying to find the relevant parameters manually, you can flag shading node parameters as important by creating an interface of exposed parameters to be used when editing the material. To expose a shading node's parameters:

1. Select the shading node and press **Alt**+**E**.

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

2. Click the menu to the right of any parameter 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 that acts as a parent for this parameter's public interface.

     If only the **Group** field is populated, the parameter's public interface becomes the actual parameter name (grouped under the contents of **Group**).

4. Once you've set the details, click **OK**.

   The parameter is exposed in a ShadingNodeSubnet's **Subnet Material Interface** and a tag appears beneath the parameter in the shading node's **Parameters** tab, listing the details of the exposed parameter.



Note: For more on the **Subnet Material Interface** and the **Material Public Interface**, continue to Collecting Shading Nodes Inside a ShadingNodeSubnet and Creating a Network Material's Public Interface.

Collecting Shading Nodes Inside a ShadingNodeSubnet

To help keep the shading network clear, 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 more in Creating a Network Material's Public Interface) of the shading nodes within.

To add nodes to a ShadingNodeSubnet node:

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

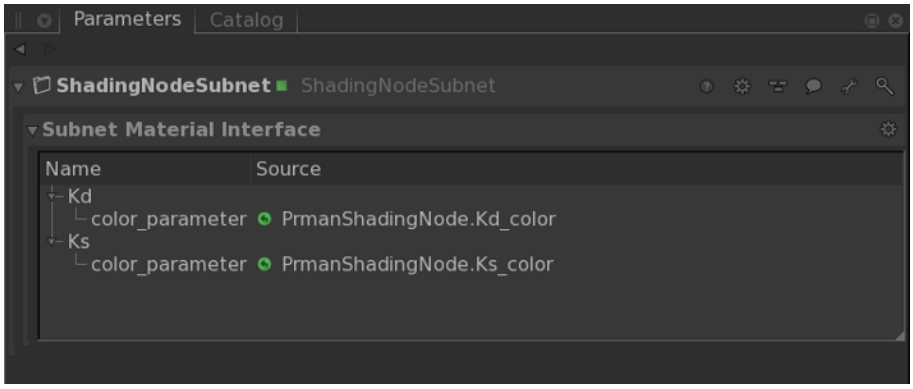
To add the shading node's exposed parameters to the ShadingNodeSubnet's **Subnet Material Interface**:

1. Select the ShadingNodeSubnet node and press **Alt**+**E**.

   The ShadingNodeSubnet node becomes editable within the **Parameters** tab and the **Subnet Material Interface** displays. By default, this appears empty.

2. Select the shading nodes with exposed parameters, and **Shift**+middle-click and drag the nodes into the **Subnet Material Interface** within the **Parameters** tab.

   The exposed parameters are shown in the **Subnet Material Interface**.



Note: If you attempt to drag any shading nodes into the **Subnet Material Interface** but they do not display, ensure that it has exposed parameters within it. If you have multiple nodes selected, and there is a node that does not have exposed parameters within it, the exposed parameters of all the selected nodes are still added to the interface.

These parameters can be re-ordered, setting a preference for how they should be displayed downstream. This preference can always be overriden by a NetworkMaterial node, and only acts as a default. To re-order the exposed parameters, middle-click and drag one group or parameter to another valid location. An orange line highlights the new position.

You can also modify the order of the exposed parameters for a NetworkMaterial node. Do this by dragging and dropping them in the **Material Interface** widget, in the base NetworkMaterial node.

Creating a Network Material's Public Interface

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 periods (**.**) 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 NetworkMaterial node's **Material Interface** parameters in the GafferThree object table's columns, see Using and Overriding Look Files with GafferThree Lights

Re-ordering the Parameters in the Network Material

The parameters with a public interface that are exposed in the network material's **Material Interface** can be re-ordered. The ShadingNodeSubnet node provides a hint as to the preferred order but, ultimately, the order is decided by the network material. To re-order the interface of the network material in the **Material Interface**, middle-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**
- **greaterThan**
- **greaterThanOrEqualTo**
- **lessThan**
- **lessThanOrEqualTo**
- **numChildrenEqualTo**
- **numChildrenGreaterThanOrEqualTo**
- **equalTo**
- **notEqualTo**
- **regex**
- **endsWith**
- **in**

• **notIn**

These tests can be combined using **and** as well as **or** logical operators. The node evaluates the test, for instance checking if the **samples** parameter is **equalTo** 0, and uses 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 the network material's scene graph location whose interface you want to control to the **materialLocation** parameter. 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 here 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

To append 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 the network material's scene graph location you want to append to the **location** parameter. For more on editing a scene graph location parameter, see Manipulating a Scene Graph Location Parameter.
3. Connect the shading network you want to append to the **append** port of the NetworkMaterialSplice node.
4. Under **inputs** > **append**, click  .

   A window appears with the current network material's shading network displays.
5. In the window, 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 the network material's scene graph location to the **location** parameter. 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**, and subsequent entries are incremented, such as **c1**, **c2**, and so forth.

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**, and subsequent entries are incremented, such as **d1**, **d2**, and so forth.
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.

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, 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.

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 the network material's scene graph location you want to edit to the **location** parameter. 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 then 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.

Handling Textures

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 to have separate materials that each explicitly declare the textures they need to use as strings 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 the 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 OpScripts.

For instance, this fragment of OpScript reads the attribute value contained in **tx_name** and use it to override a shader with a parameter called **textureFile**:

```
tx_name_attr = Interface.GetAttr("tx_name")
if tx_name_attr then Interface.SetAttr
("materialOverride.parameters.textureFile", tx_name_attr)
end
```

This means that a new copy of the material is created for the object with the shader's **textureFile** parameter changed to the appropriate values.



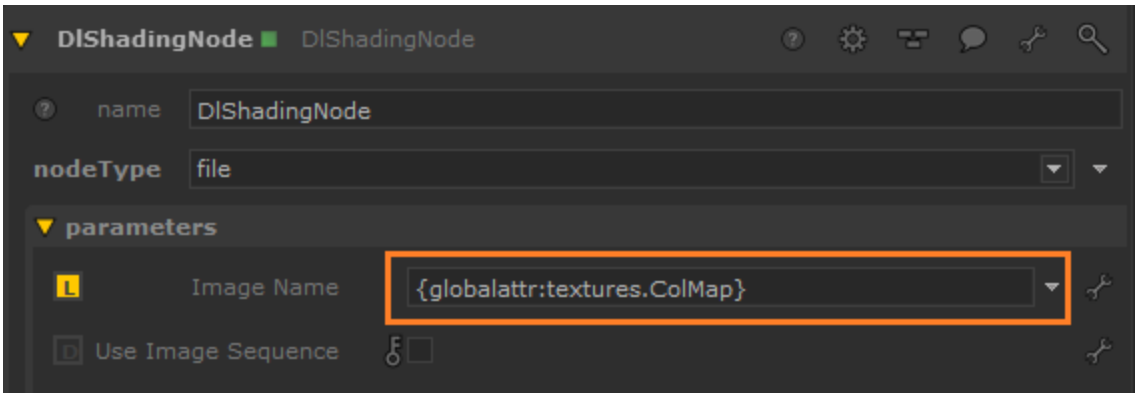
Note: Material overriding actually takes place as part of MaterialResolve, one of Katana's implicit resolvers. During MaterialResolve it looks for attributes in the 'materialOverride' group, and creates a new copy of the material at that location with the relevant changed to shader parameters.

Using the {attr:xxx} and {globalattr:xxx} Syntax for Shader Parameters

For versatility, shader parameters can be set to the value of an attribute. You can do this using either local or global attributes:

{attr:<attribute name>} - queries local attributes

{globalattr:<attribute name>} - queries global (inherited) attributes



Note: Querying global attributes has a higher processing cost than local attributes.

If you define any string parameter on a shader to be **{attr:xxx}** (for example), then during MaterialResolve it looks for an attribute called **xxx** at the location the material is being assigned to, and uses that as the shader value.

To illustrate, suppose you have a 3Delight file reader shader with a parameter called **filename**, and you set **filename** to **{attr:textures.ColMap}**. **filename** is set to the value of the attribute **textures.ColMap** 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.

Example of Using {attr:xxx} to Assign a Parameter

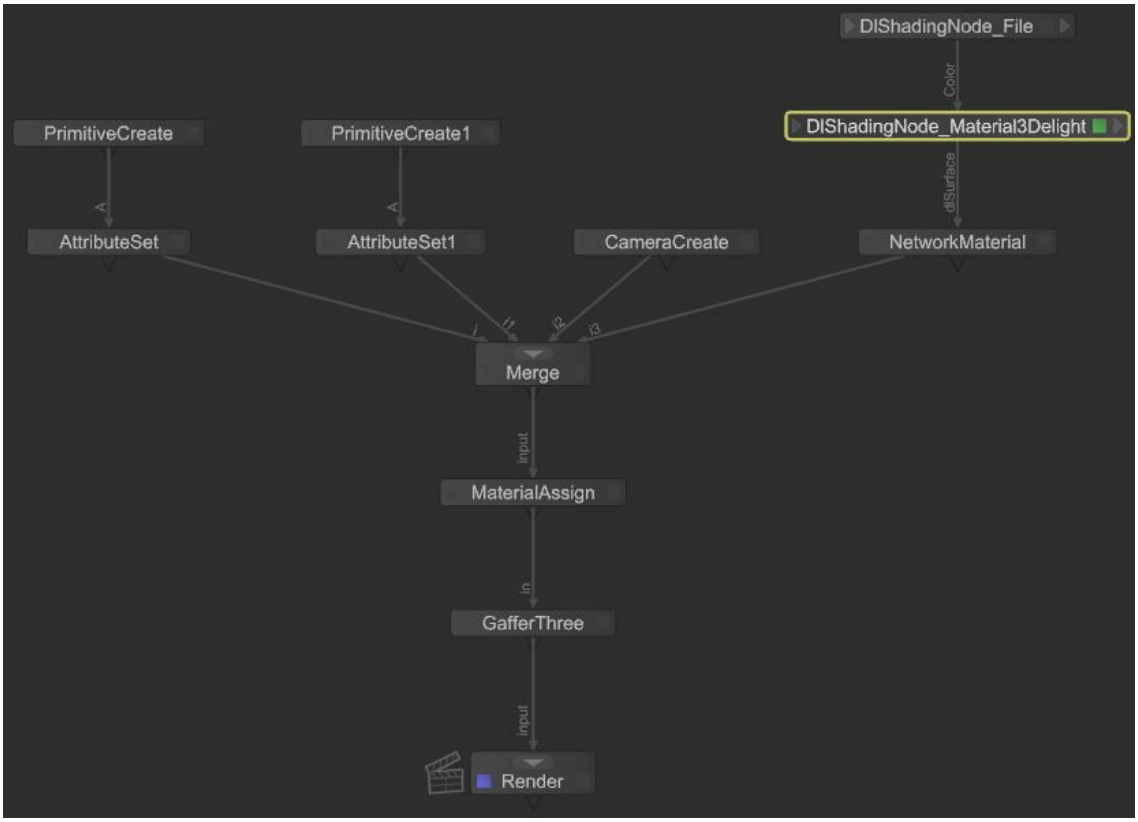
The following steps show you how to create a 3Delight Network Material with a File node, and link the filename parameter of the ImageRead node to a **textures.ColMap** attribute on geometry locations.

1. In an empty Katana scene create two DlShadingNodes, set one to type **File**, and one to type **Material3Delight**. Link the **outColor** of the **File** node, to the **BaseLayer>Diffuse>Color** parameter of the **Material3Delight** node.
2. In the **filename** field in the Image node's parameters enter:
   `{attr:textures.ColMap}`
3. Add a NetworkMaterial node, with a **dlSurface** terminal. Connect the output of the **Material3Delight** node 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 GafferThree below the RenderSettings node, and add a spotlight.
8. Add a Render node below the GafferThree node.
9. Position the spotlight, and the camera.
10. 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 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 3Delight and 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 the renderer 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.

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 example project **Crowd System - Look Dev & Metadata** illustrates how metadata can be further processed by an OpScript to turn it into final texture paths. By setting these in attributes called **textures.ColMap**, **textures.SpecMap** and **textures.BumpMap** these are exported to renderers as "MatTag" attributes.

Checking UVs

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

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.

To zoom in, move your mouse pointer over the area you want to zoom in on, and:

- 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.

To zoom 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

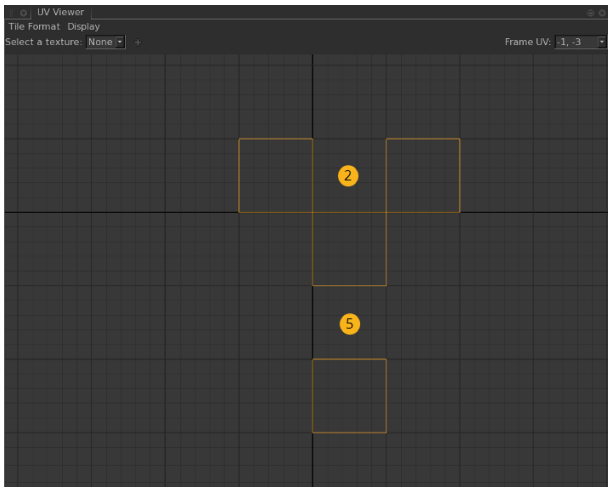
To change the framing within the **UV Viewer**, you can:

- 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. 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.





You can see in the two images above that, of the six faces originally selected, faces two and five have been de-selected in the second image.

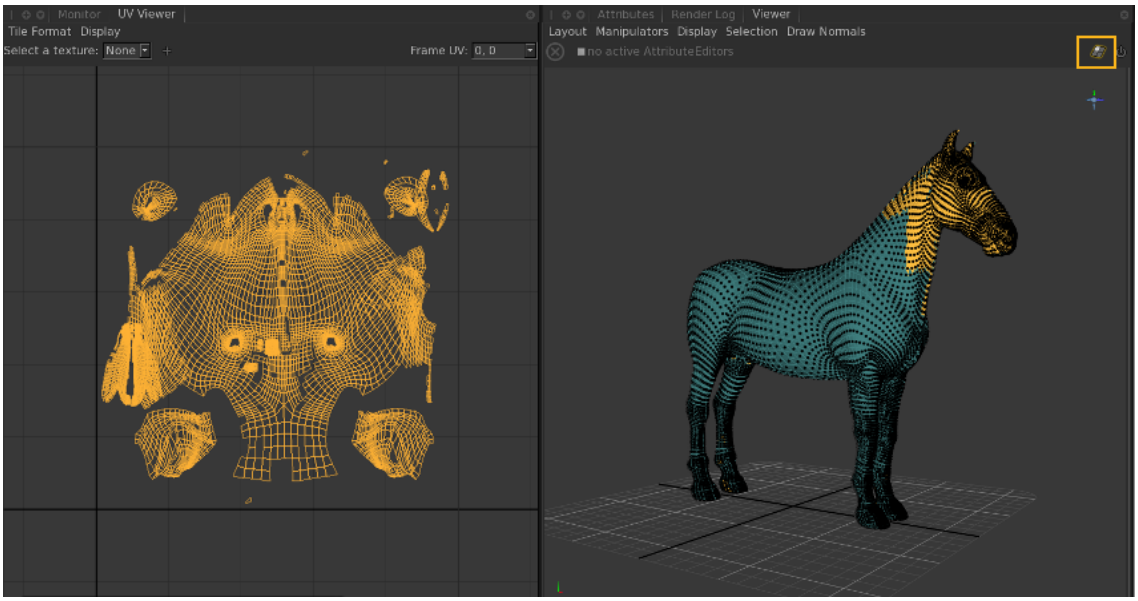
Modifying an Existing Selection

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  .



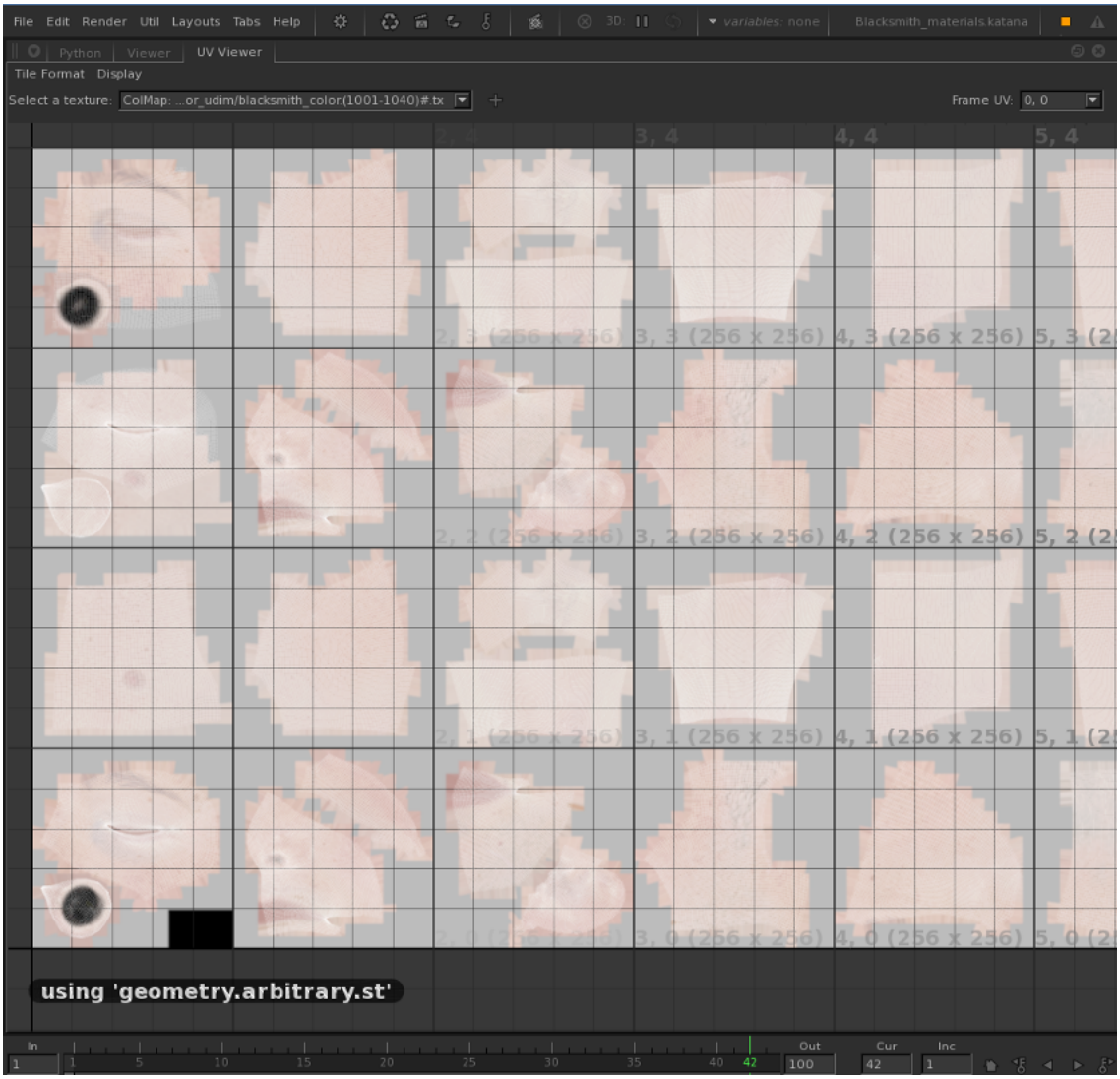
Adding Textures to the UV Viewer

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**.

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™.

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 the asset management and the file sequence plug-in, see the included technical PDFs in the **${KATANA_ROOT}/docs/pdf** directory.

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.

To toggle whether selected faces are shown in isolation select **Display** > **Isolate Selection**. If active, when faces are selected all other faces become hidden.

To change the labels for each tile, select **Display** > **Tile Position Labels** > **...**. The following options are available:

- **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.

To show the texture resolution for any textures, select **Display** > **Show Texture Resolution**.

To only display textures when contained within a face, select **Display** > **Clip Textures to Poly**.

Look Files

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, 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.

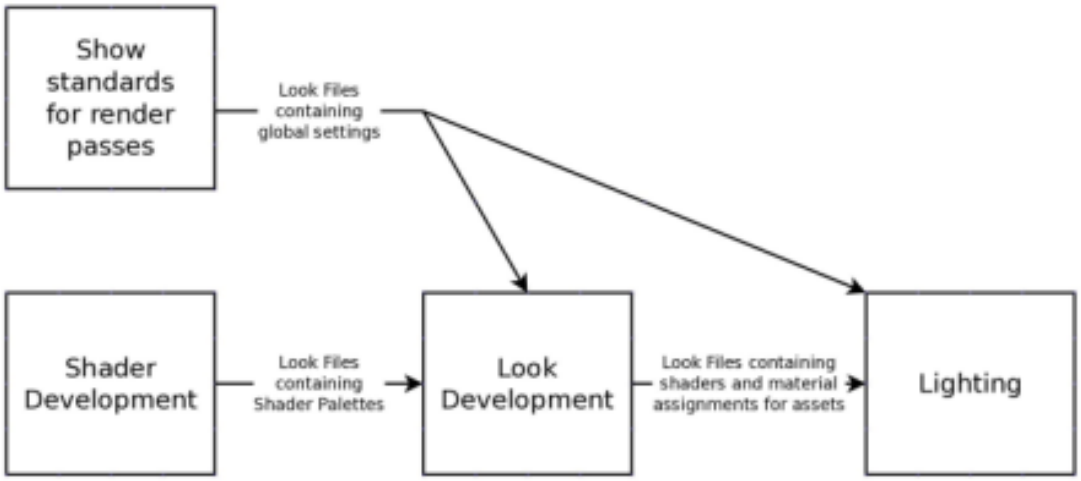
If a new version of the asset is created, any associated Look Files need to be baked out again by re-running the LookFileBake in the appropriate Katana project.

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**.

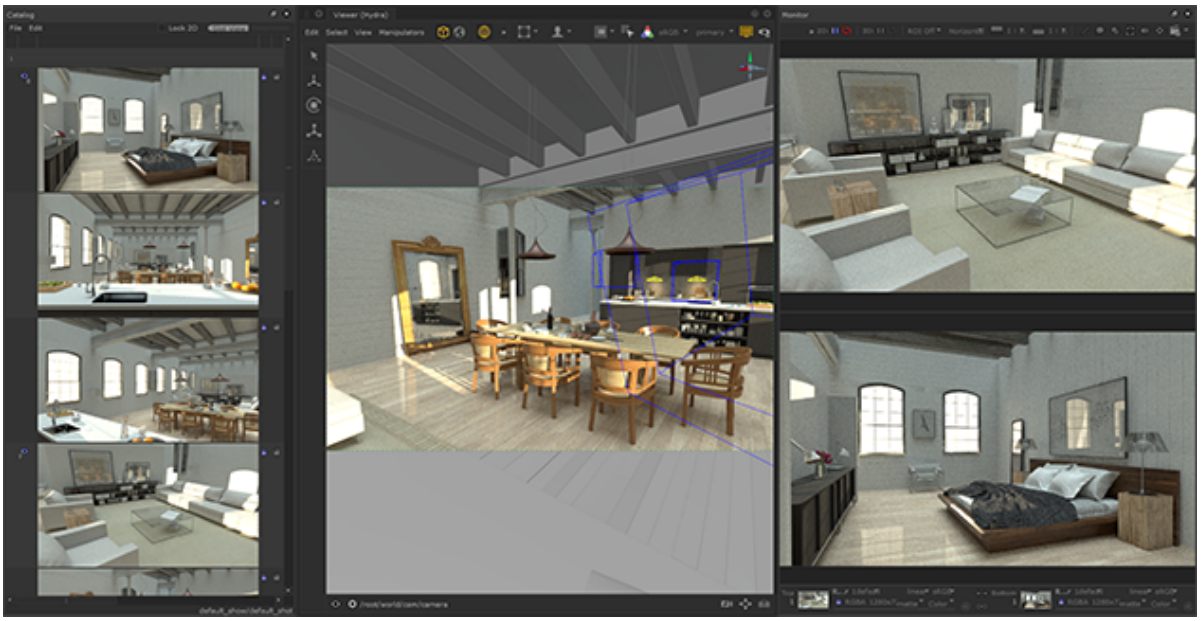
The Look File Manager

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 SuperTool 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.

Rendering Your Scene

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 the 3Delight renderer and accompanying renderer plugin, and an API that allows developers to support other renderers.

Scalability is also at the heart of rendering in Katana. Many renderers support procedurals that can be evaluated on demand. These are 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** 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

Discover Katana's render options including Preview Rendering, Live Rendering, Disk Rendering, and rendering using Render Farms.

Performing a Render

Learn how to start each render type and in Katana and integrate them into your workflow.

Configuring a Render

Learn how to configure your renders to suit your requirements.

Viewing Your Renders

Find out how to view your renders using the Monitor, Catalog or Hydra Viewer tabs.

Custom Render Resolutions

Discover how to define custom render resolutions to supplement or replace Katana's pre-defined resolutions.

Influencing a Render

Learn what key nodes you can use to influence your renders by setting the render camera, resolution, render outputs and more.

Controlling Live Rendering

Discover how you can control Live Rendering behavior.

Setting up a Render Pass

Learn how to set up render passes in Katana using the RenderOutputDefine node.

Instancing

Find out how instancing can help you when needing to render a single piece of geometry multiple times in a scene.

OpenEXR Header Metadata

Learn how you can add arbitrary metadata to OpenEXR headers.

Batch Mode

Discover how to batch render sequences of frames from a Katana scene all at once through a command line.

Render Types

Katana has a number of context-sensitive render options, available through the right-click menu on nodes. Renderer plug-ins advertise the methods they support, so the right-click menu render options shown depend on the node selected and the methods advertised in the renderer plug-in.



Note: For more information about which options are available from which nodes, see Render Type Availability.
For more information about how to start a render, see Performing a Render.

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, **Monitor Layer** and **Catalog** tab. The image is not written to disk.

Live Rendering

- **Live Render** - Similar to **Preview Render**, except that under **Live Render**, changes to the camera, lights, materials, or geometry transformations result in updates to the image displayed in the **Monitor** tab **Monitor Layer** and **Catalog** tab. See Changing How to Trigger a Live Render for more on which activities trigger a **Live Render**, and how to edit them.



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 current render, including Live Rendering, either press **Esc** or select **Render** > **Cancel Current Render** in the menu bar. To stop all renders, press **Shift** + **Esc** or select **Render** > **Cancel All Renders**.

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. It is possible to perform multiple Live Renders simultaneously using the experimental option in the **Start Multiple Renders** script for the **Katana Queue**. For more information, see Katana Queue.

Profile Rendering

- **Preview Render with Profiling** - This performs a normal **Preview Render**, but also captures information about which Ops have run, the amount of CPU used by them to cook locations, and the amount of memory used for attributes and Lua scripts.



Note: For more information, see Geolib3-MT Profiling.

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.
- **Render Dependencies Only** - renders just dependencies to disk.

Render Farms

Any render farm plugins that you have set up in Katana can be accessed here from any 3D node. Katana ships with a render queue system called Katana Queue.

- **Katana Queue** - Send a **Preview**, **Live** or **Disk Render** to the Katana Queue. Renders sent to Katana Queue can be viewed in the **Katana Queue** tab.



Note: For more information, see Katana Queue.

Disk Render Dependencies

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** that holds the following options:

- **Before Preview Renders** - when selected, render dependencies (such as shadow maps) are rendered to disk before performing a **Preview Render**.
- **Before Live Renders** - when selected, render dependencies are rendered to disk before **Live Rendering**.
- **Before Profiling Renders** - when selected, render dependencies are rendered to disk before **Profile 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** that 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.

Debugging

3D nodes have a right-click menu sub-heading **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.

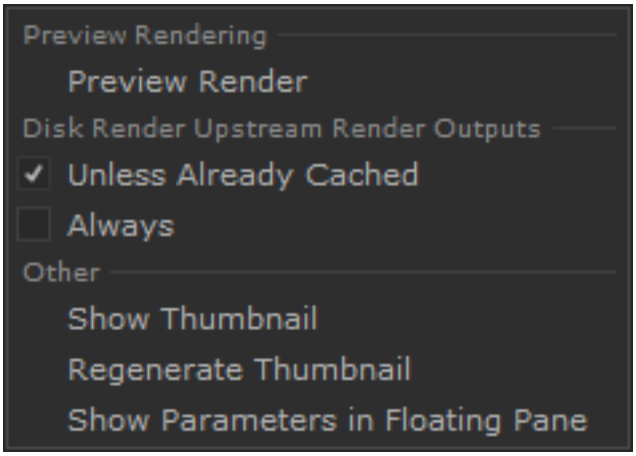
Render Type Availability

This topic lists the available render options from different node types. For more information about the render options, see Render Types.

2D Image Nodes

For 2D image nodes (such as ImageGamma or ImageCrop) the right-click menu render options are:

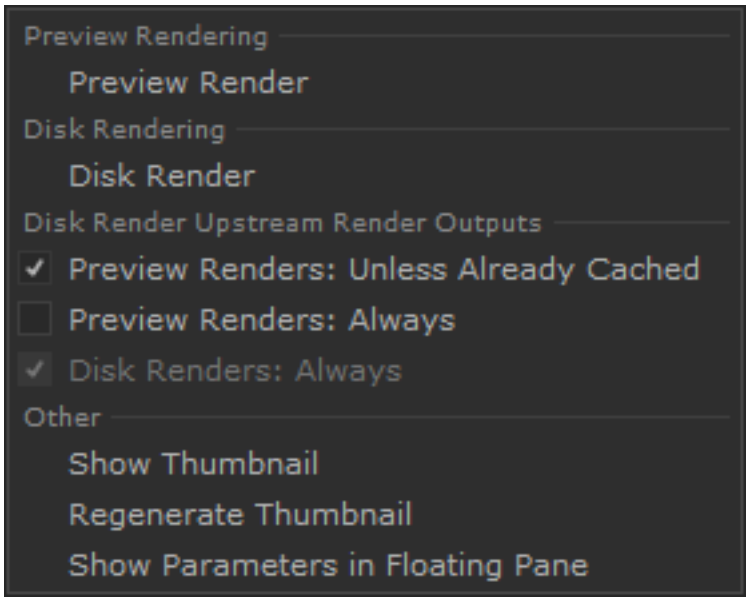
- **Preview Render**
- **Unless Already Cached** (specific to disk rendering upstream)
- **Always** (specific to disk rendering upstream)



2D ImageWrite Nodes

The render types available from an ImageWrite node are:

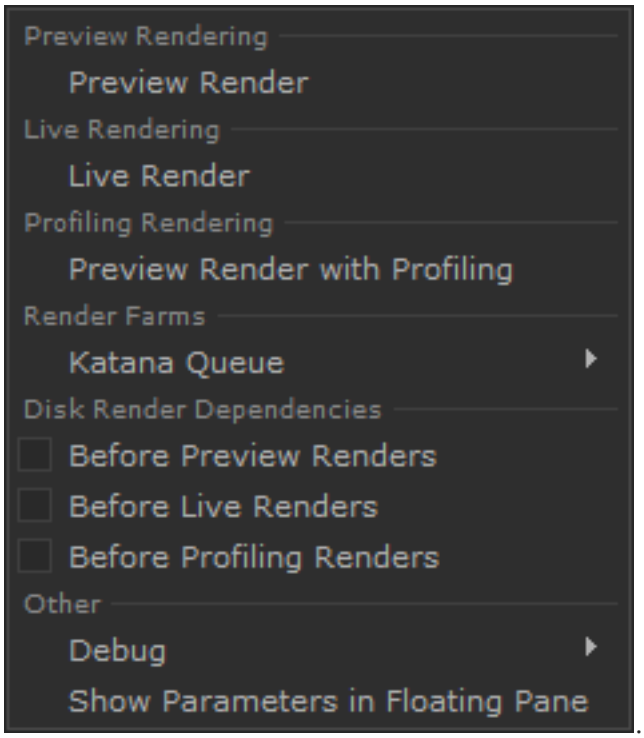
- **Preview Render**
- **Disk Render**
- **Preview Renders: Unless Already Cached** (specific to disk rendering upstream)
- **Preview Renders: Always** (specific to disk rendering upstream)
- **Disk Renders: Always** (specific to disk rendering upstream)



3D Nodes

The render types available from 3D nodes (such as a PrimitiveCreate, CameraCreate, or Material nodes) are:

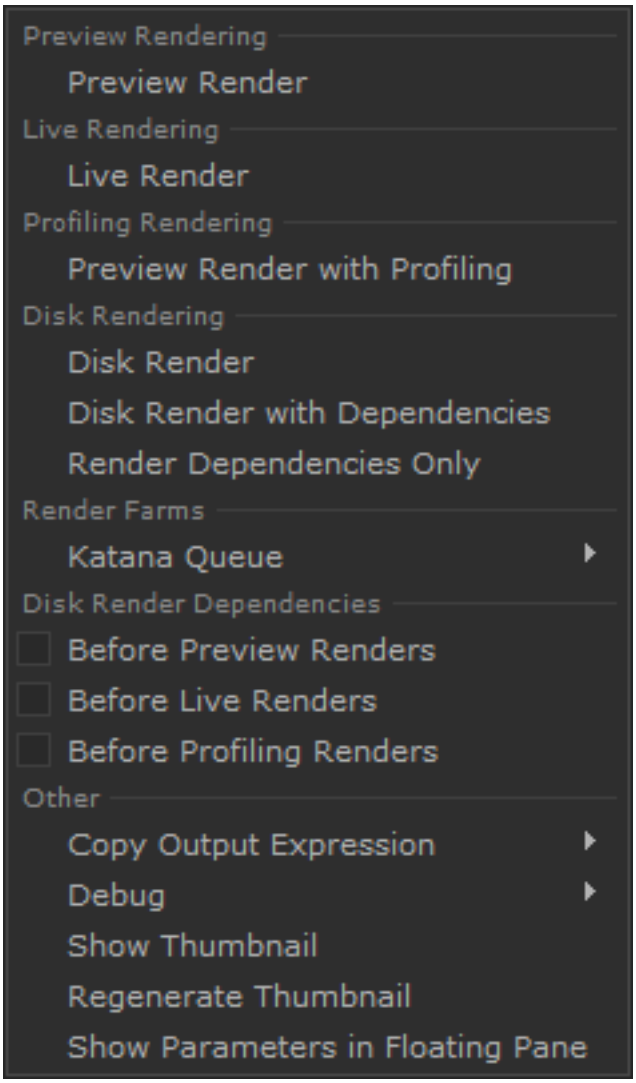
- **Preview Render**
- **Live Render**
- **Preview Render with Profiling**
- **Katana Queue**
- **Before Preview Renders**
- **Before Live Renders**
- **Before Profiling Renders**



Render Node

The render types available from Render nodes are:

- **Preview Render**
- **Live Render**
- **Preview Render with Profiling**
- **Disk Render**
- **Disk Render with Dependencies**
- **Render Dependencies Only**
- **Katana Queue**
- **Before Preview Renders**
- **Before Live Renders**
- **Before Profiling Renders**



Performing a Render

You can start a render from the Katana UI by clicking **Render** and choosing a render option, or by right-clicking a node and choosing a render option. Bear in mind that not all options are available from all nodes.

Note: For more information about the render options, see Render Types.
For more information about which options are available from which node types, see Render Type Availability.

To cancel the current render, press **Esc**, or select **Render** > **Cancel Current Render**.

To cancel all renders, press **Shift** + **Esc** on the keyboard or select **Render** > **Cancel All Renders**.

You can repeat the previous render by pressing **Ctrl** + **** (**backslash**), or choosing **Render** > **Repeat Previous Render**.



Note: You can also perform Preview Renders, Live Renders and Disk Renders using the **Katana Queue** options, for more information on the **Katana Queue**, see Katana Queue.

Performing a Preview Render

You can perform a **Preview Render** at any 3D node within your recipe. A scene description is generated up to that node. The extent that 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, the Catalog tab and the Monitor Layer in the Hydra Viewer.

To perform a **Preview Render**:

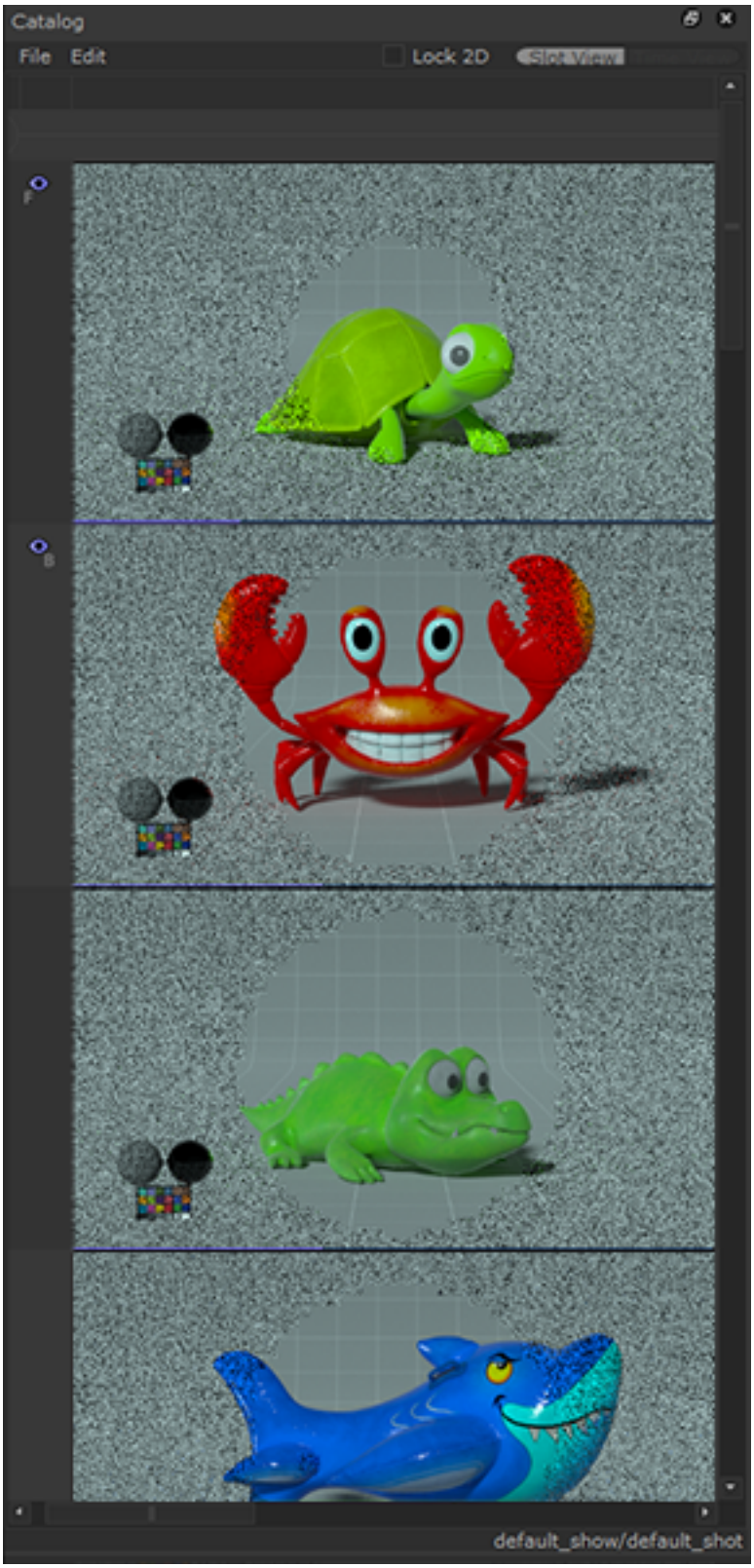
1. Right-click on a 3D node within your **Node Graph**.
2. Select **Preview Render**.

You can also start a **Preview Render** from a node currently set with the view flag by selecting **Render** > **Preview Render View Node**, or by pressing **Ctrl** + **P** on the keyboard.

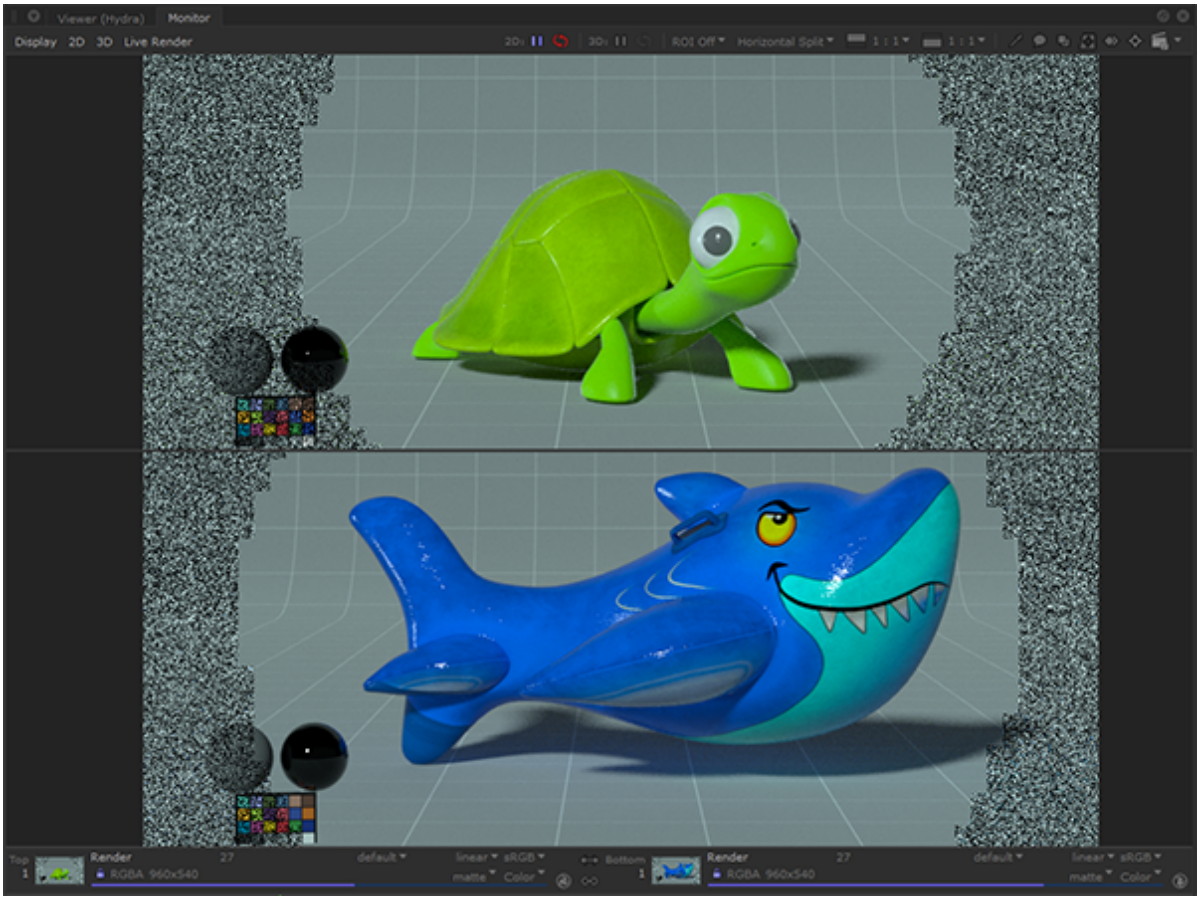
Performing Multiple Simultaneous Preview Renders

Multiple **Preview Renders** can be performed simultaneously in Katana. By default, if a **Preview Render** is already running and you start another, both **Preview Renders** continue until complete or canceled. Katana does not limit the number of **Preview Renders** you are able to run simultaneously, you are only limited by hardware capabilities.

The ability to run multiple **Preview Renders** simultaneously is useful as it means you don't need to wait until one render is finished before starting a new one. This can help workflows for lighting artists, for example, as multiple camera angles can be rendered at once, streamlining the process of ensuring lighting is consistent across different shots.



Multiple **Preview Renders** in the **Catalog** tab



Multiple **Preview Renders** in the **Monitor** tab

Note: You can start multiple Preview Renders by using the **Start Multiple Renders** dialog that you can find in the **KatanaQueue** shelf. For more information on the **Katana Queue**, see Katana Queue.

Article: If you are experiencing issues when rendering in Katana, please refer to the Knowledge Base article: Troubleshooting Rendering Issues in Katana.

Article: If you are using Linux and are experiencing issues where your render hangs, please refer to the Knowledge Base article: Render Hangs or Never Starts on Linux.

Performing a Live Render

Live Renders are useful for getting immediate feedback on changes you make to objects, 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.

To start a Live Render:

1. In the **Scene Graph** tab, check the objects in the **Live Render Updates** column  that you would like to be able to trigger a Live Render to restart when changes are made to those objects.
2. Right-click on the node that you want to start a Live Render from and select **Live Render**. You can also start a **Live Render** from a node currently set with the view flag by selecting **Render** > **Live Render View Node**, or by pressing **Ctrl** + **Shift** + **P** on the keyboard.
3. Adjust the parameters of any object in your **Scene Graph** that has been included for **Live Render Updates**.

   The image in the **Monitor** tab, **Catalog** tab and **Monitor Layer** updates in response to your actions. 3D node parameter values are finalized with all pending changes prior to performing a render.

See Starting Multiple Renders and Multiple Live Renders with Foresight+ for more information on performing multiple Live Renders simultaneously.



Article: If you are experiencing issues when rendering in Katana, please refer to the Knowledge Base article: Troubleshooting Rendering Issues in Katana.



Article: If you are using Linux and are experiencing issues where your render hangs, please refer to the Knowledge Base article: Render Hangs or Never Starts on Linux.

Preview and Live Rendering with Nuke Bridge

Nuke Bridge allows lighting and look development artists to see their work in the context of a Nuke render. You can stream your Katana **Preview** and **Live** renders to Nuke running either on your local machine or sitting on the render farm.

Nuke Bridge offers three modes of operation depending on how you want to work with your render and composite:

- **Preview Comp mode** - Sends a Katana render to Nuke, then streams a comp back for a quick and easy snapshot of the render in the composite. This mode runs Nuke as a background process (locally or remotely).
- **Live Comp mode** - Allows you to make changes in the Katana project, such as change the render fed into the Nuke Input Points panel in the Nuke Bridge tab, and see those changes come back from Nuke. This mode also runs Nuke as a background process (locally or remotely).

  When used together with live rendering, you can adjust scene properties such as lighting, materials or cameras and receive the updated render back through Nuke.
- **Interactive Comp mode** - Launches Nuke so you can make edits to both your Katana scene and Nuke script simultaneously.

See See a Nuke Comp of Your Project in Katana Using the Nuke Bridge for more information.

Executing a Disk Render

Disk Renders can only be performed from a Render node. Render nodes acts as a render point within a Katana 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.

   Tip: Add a RenderOutputDefine node above the Render node to define the output name, format, and file location.

2. 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.



Warning: Progressive interactive renders, when configured to send image updates (buckets) with high frequency, may flood the message queue of the renderer plug-in's display driver. To prevent this from consuming unreasonable amounts of memory, the queue is limited in size and, when full, results in delays in updates being sent to Katana. A warning is then printed to the Render Log. The size of the queue (as a number of messages) can be specified using the environment variable **KATANA_PIPE_MAX_QUEUE_SIZE**. The default size is 16384.

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
  ```

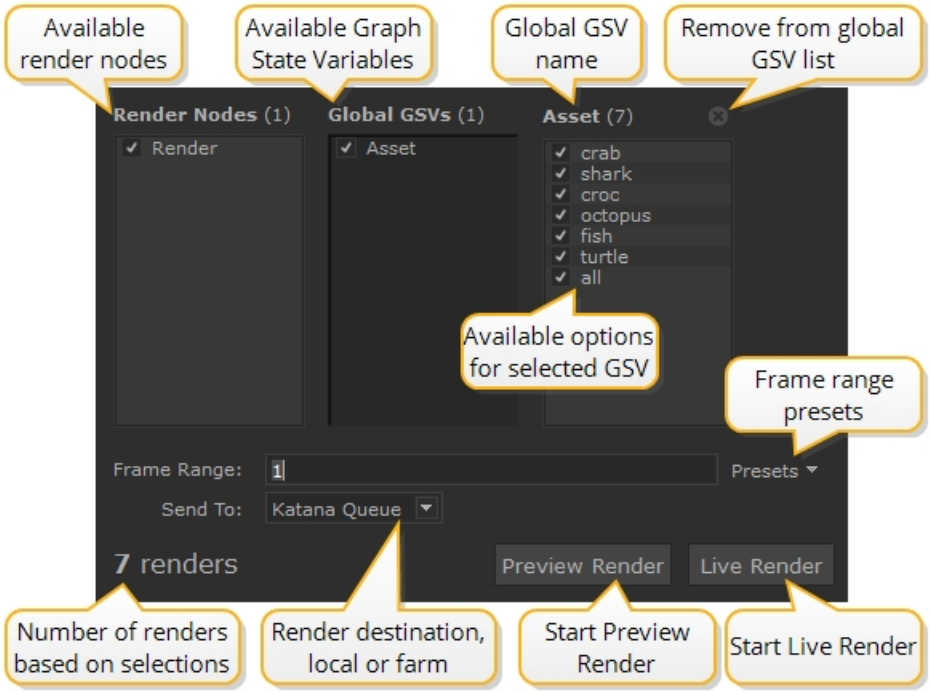
- **Script** mode - This mode allows you to execute a Python script in Katana's Python environment, so you can perform more complex actions such as changing parameters, creating nodes, or modifying node connections, as well as launching renders.



Note: For more information on how to use **Batch** and **Script** modes, see Command-line Interface.

Starting Multiple Renders

The **Start Multiple Renders** dialog allows users to start multiple Preview or Live Renders by selecting various combinations of global Graph State Variables (GSV). The ability to start multiple renders at once and have them render simultaneously can greatly improve efficiency as you don't have to wait for one render to finish before starting another. This can be very useful to compare lighting adjustments across multiple shots, or material and asset variations, for example.

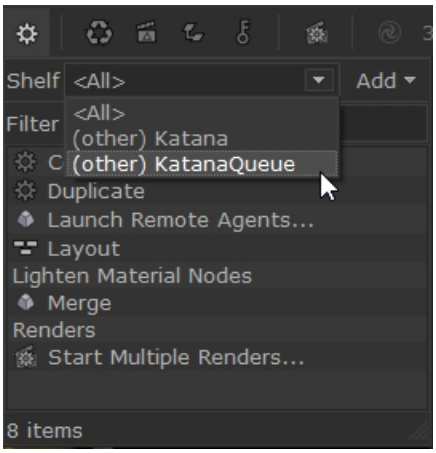


Note: Multiple simultaneous Live Renders are not allowed by default. Starting a new Live Render cancels any currently active Live Render. To enable simultaneous Live Renders, go to **Preferences** > **application** > **rendering** and enable **allowConcurrentRenders**.

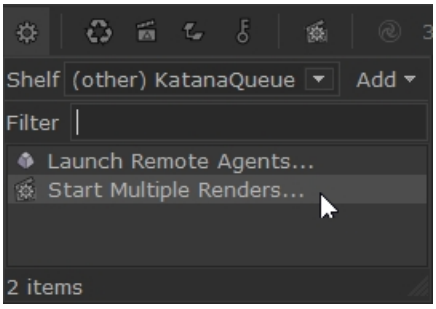
Using the Start Multiple Renders Dialog

To use the **Start Multiple Renders** dialog:

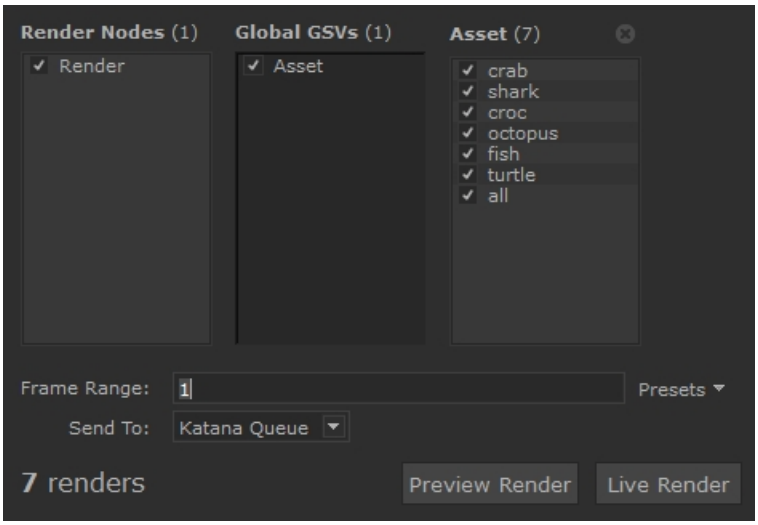
1. Click the **Shelf Actions** button at the top of the Katana UI.
2. Select **(Other) KatanaQueue** from the **Shelf** dropdown.



3. Select **Start Multiple Renders...**



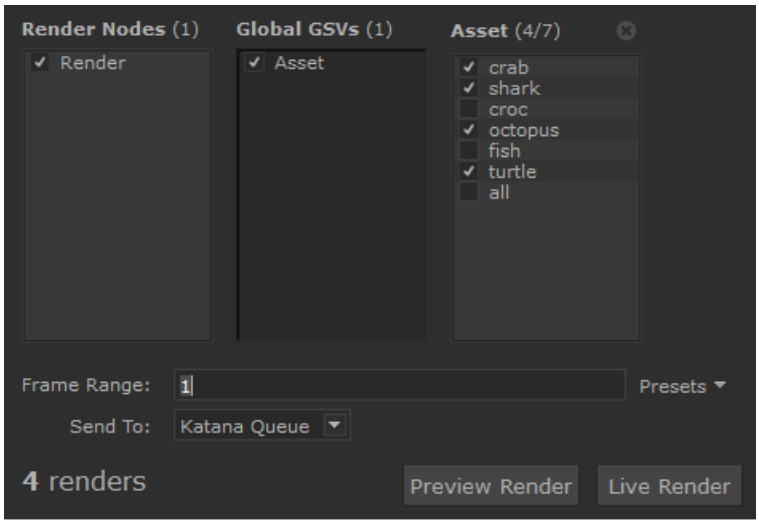
The **Start Multiple Renders** dialog opens.



4. Choose the **Render Nodes**, **Global GSVs**, and individual GSV options you want to render.

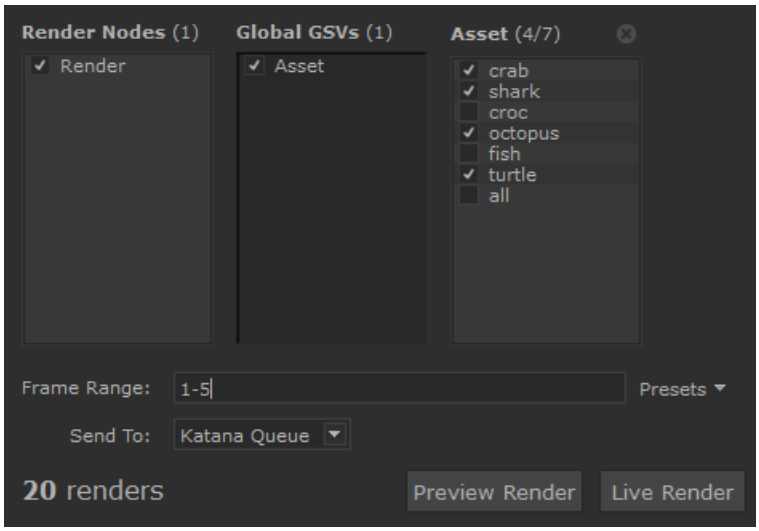
   All available options are selected by default.

The total number of renders that result from the current selections of **Render** nodes and **GSVs** is shown and updated in a label at the bottom left of the dialog.

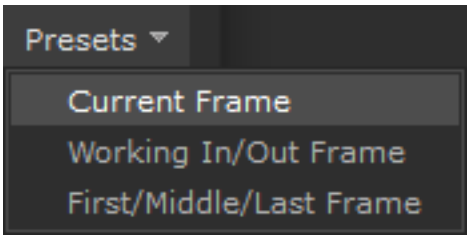


5. Enter the frames you want to render in the **Frame Range** text field.

   You can enter individual frames, or frame ranges by separating them with commas, for example: 1-10, 20, 30-45.

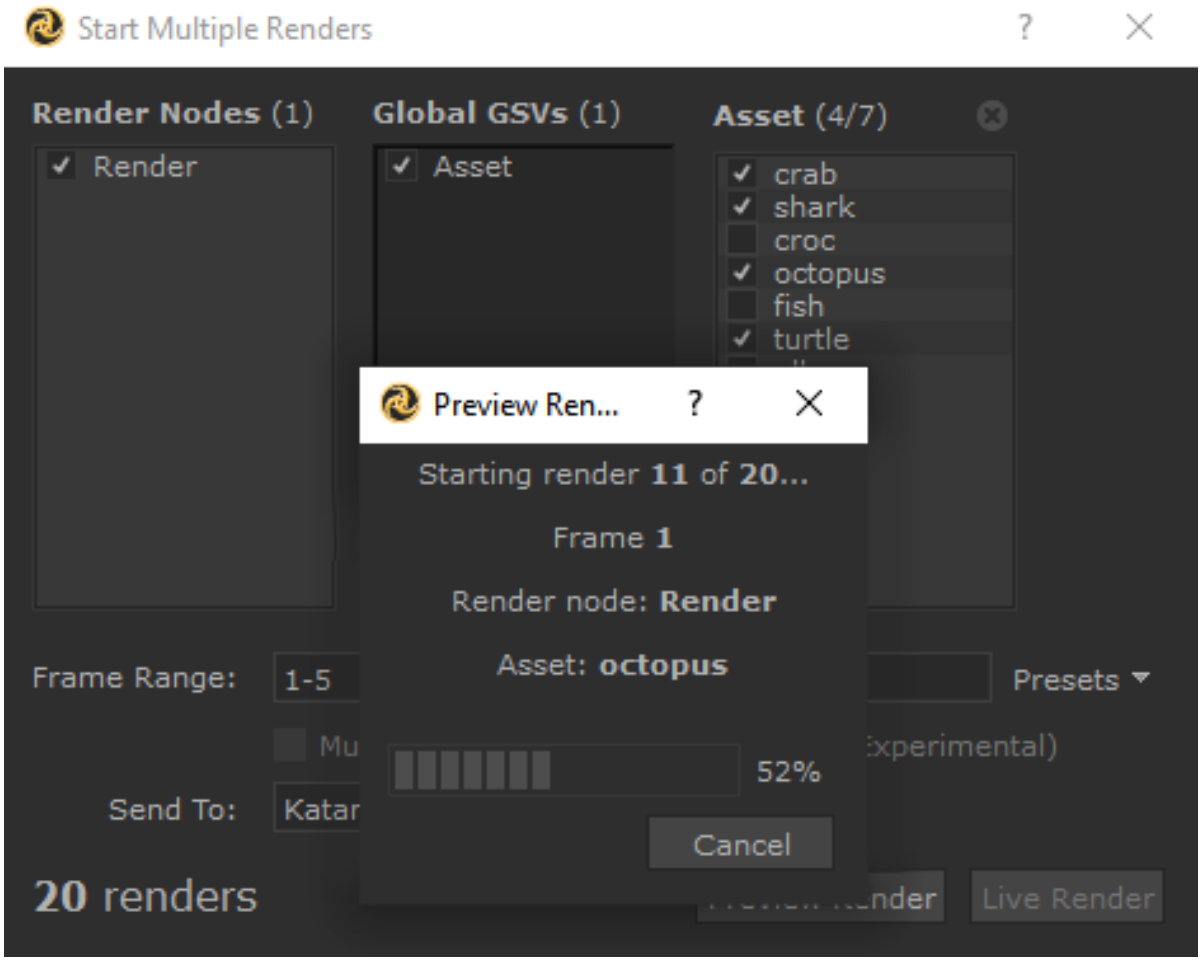


You can also use the **Presets** dropdown to select from **Current Frame**, **Working In/Out Frame** and **First/Middle/Last Frame** of the timeline of the current Katana project.

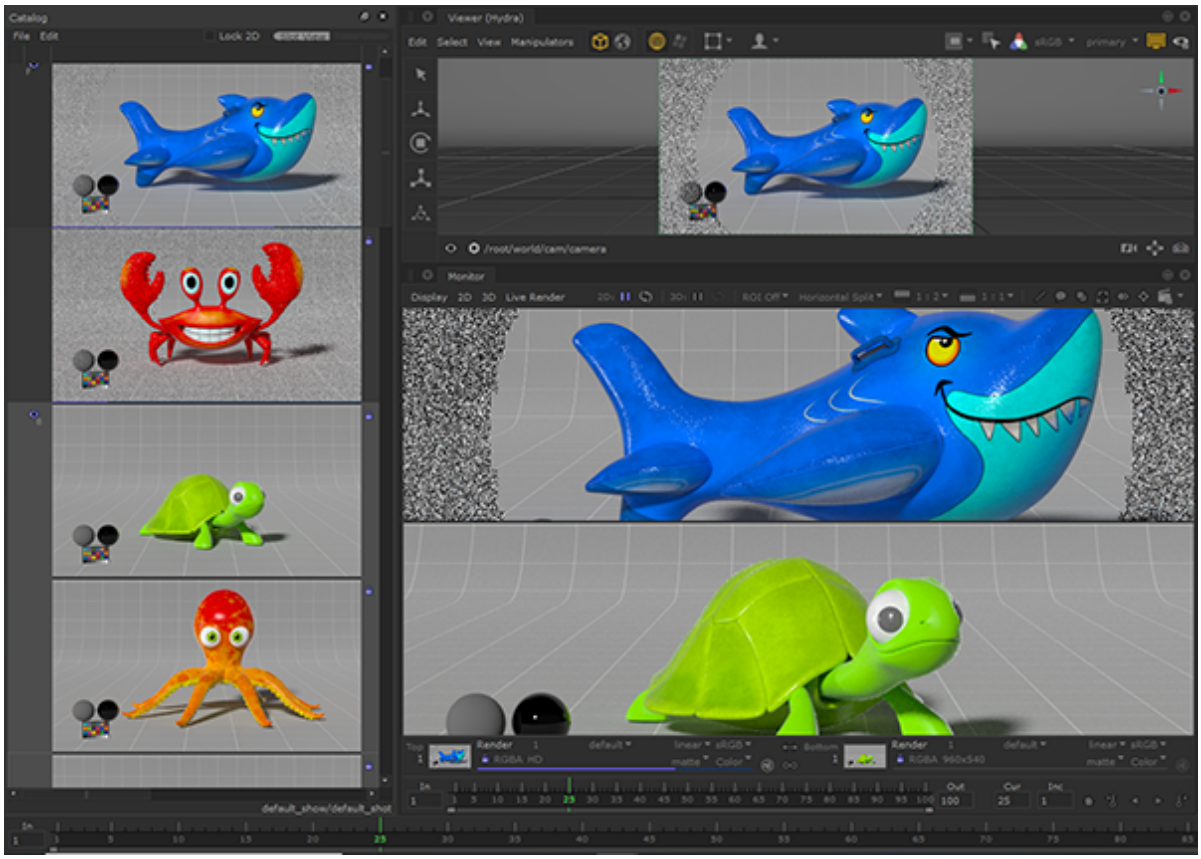


6. In the **Send To** dropdown menu, choose the render farm you want to send the render jobs to or choose Local Machine to use your local machine to perform the renders.
7. Click **Preview Render** or **Live Render**.

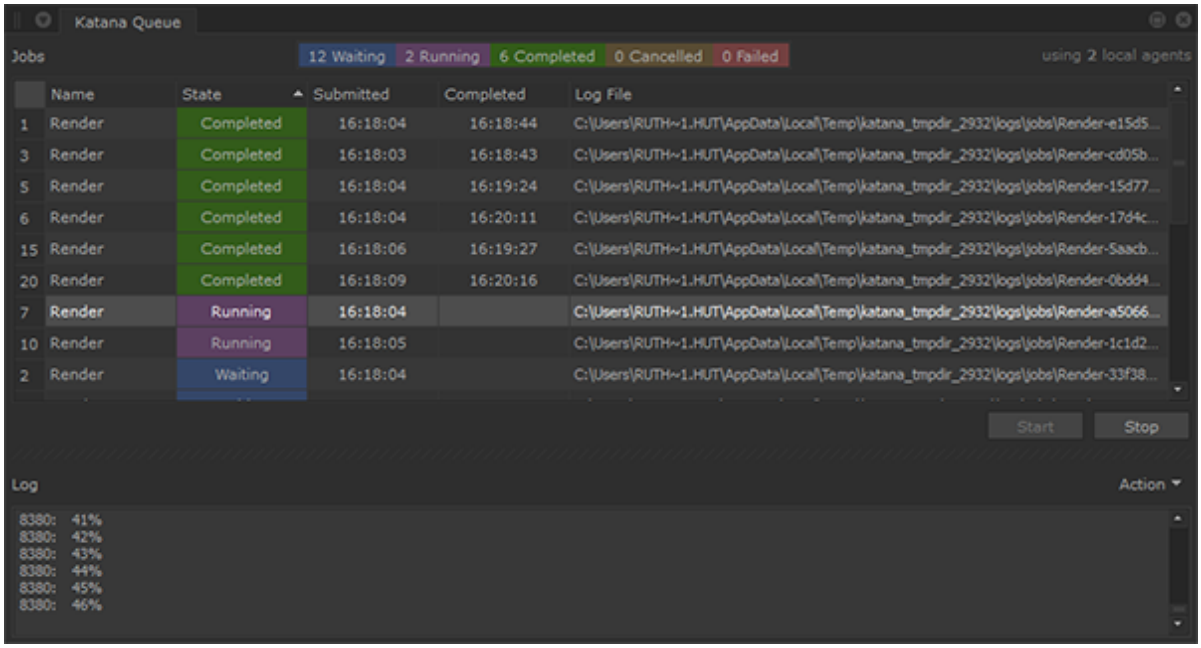
   A **Preview Render** or **Live Render** dialog appears while the renders are starting.



The images begin rendering and can be viewed in the **Catalog** tab, the **Monitor** tab and the **Monitor Layer** in the **Hydra Viewer**.



Render jobs are displayed in the **Katana Queue** tab.



See Multiple Live Renders with Foresight+ for more information on Live Renders.

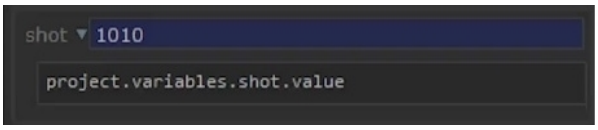
Multiple Live Renders with Foresight+

Foresight+ gives you the ability to dispatch multiple Live Renders to any machine, whether local or on the farm using Katana Queue. You can work on as many shots as you like from one Katana scene and see every change you make that influences all the shots in a sequence, live, inside Katana. Working on shots in a sequence concurrently raises the quality of all shots simultaneously and helps to avoid surprises when lighting conditions change.

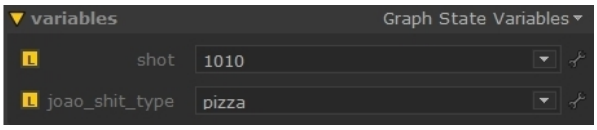
Foresight+ includes a significant restructuring of how live rendering is handled inside Katana. As part of this restructure, attribute changes are now communicated to individual render processes, meaning that any changes only trigger updates for the affected renders. This means your shot-specific lighting or look-dev changes only trigger updates for the affected live renders so you don't need to worry about inaccuracies or being slowed down by unnecessary render updates.



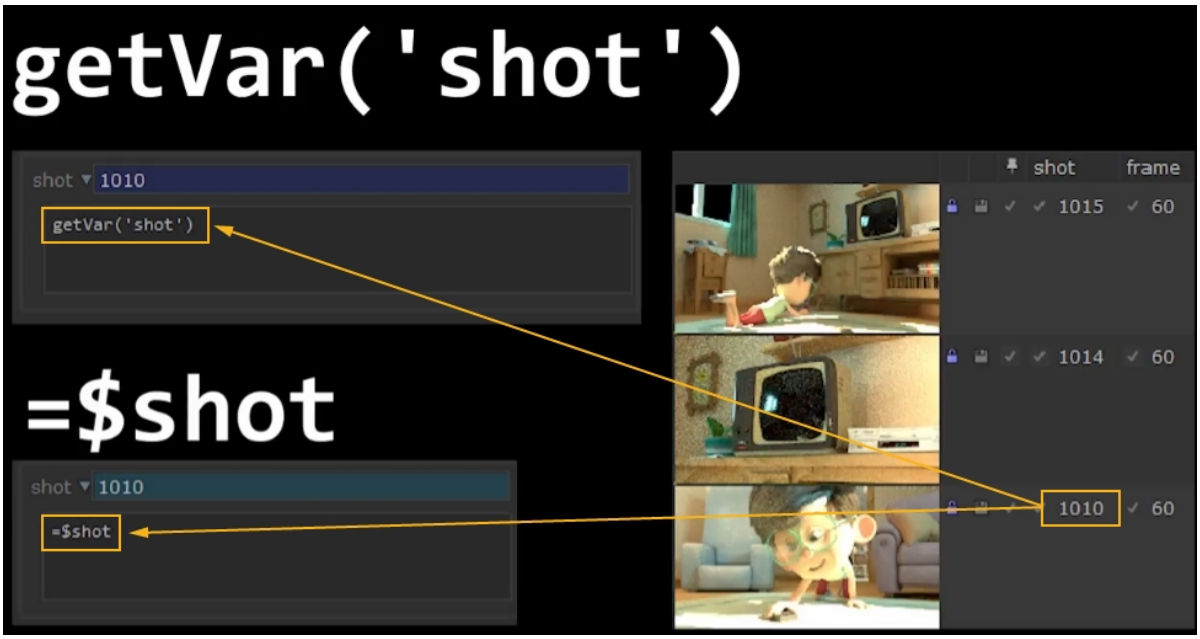
Instead of accessing Graph State Variable (GSV) values from the parameters inside the **Project Settings**, the new expressions grab the values from **Catalog** items.



The old **project.variable.shot.value** variable



The new **getVar('shot')** variable that references Catalog items



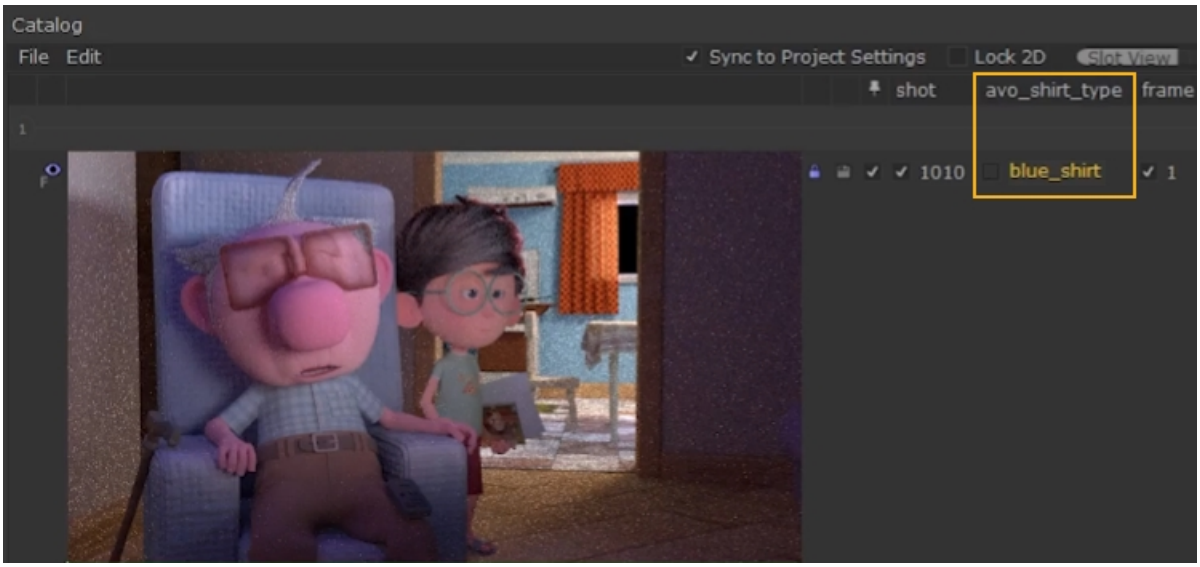
This means that for each render, the expressions are evaluated to the values in the **Catalog** items, rather than GSV values. As a result, switching between different GSVs and scrubbing through the timeline doesn't trigger a live render update unless you want it to.

See Graph State Variables for more information on setting GSVs and Using the Catalog Tab for more details on the **Catalog** tab.

Pinning Shots in the Catalog to Control Live Renders

Pinning renders in the **Catalog** tab allows you to control what changes in the scene trigger an update to Live Renders currently in the catalog. When values are pinned, any changes made to the GSVs or current frame won't trigger an update. Once a parameter is unpinned from a **Catalog** item, the value can be overridden by making changes to the global settings. Once changed, the unpinned value turns yellow to indicate an override.

In this example, unpinning the **avo_shirt_type** variable, shown in yellow, means that any change to that variable triggers a Live Render update. Changing the **shot** or **frame** does not update the **Catalog** for this pinned entry.



A live render catalog render that only updates when the **avo_shirt_type** variable is changed

Note: All parameters in the **Catalog** tab are unpinned unless the renders were started using the **Start Multiple Renders** dialog. See Starting Multiple Renders for more information. Pinning can be enabled by default in the **Preferences** under **application** > **rendering** > **renderVariablesPinnedByDefault**.

You can choose to unpin all parameters using the tickbox at the front, or make individual choices for each parameter. This is a really useful feature as you may want to switch between a few different shots for one live render, while the others remain on a specific shot. Similarly, you may want you scrub through the timeline for one specific shot and ensure that your lighting is consistent across its entire length. This may be unnecessary for other shots that don't involve as much range.

Overriding Global GSVs with Catalog Entries

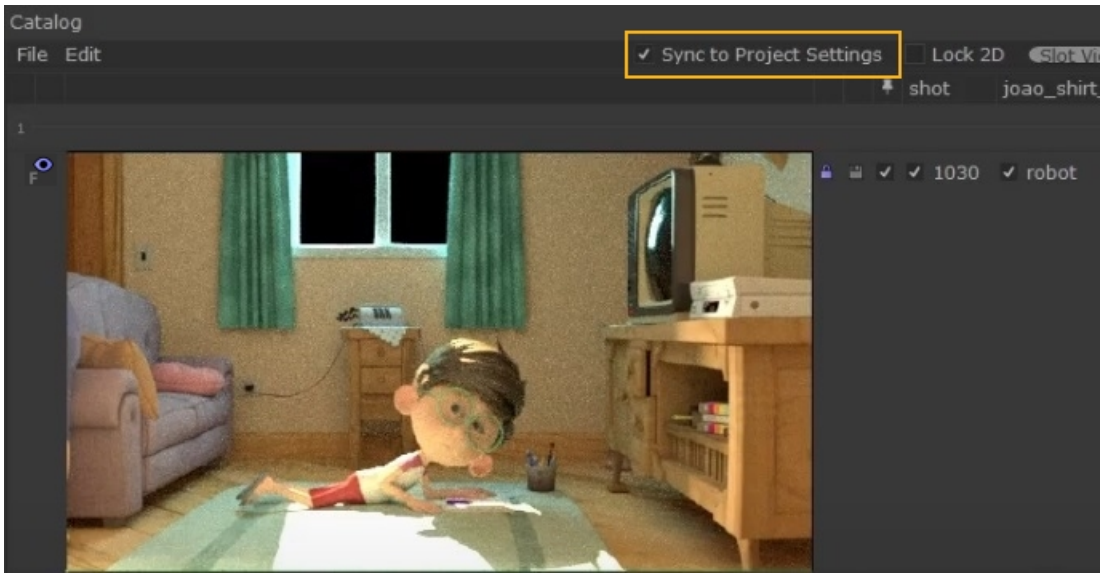
You can work with multiple shots and avoid mismatches between the geometry in the Viewer and your live renders by syncronizing the global GSVs in the **Project Settings** with a selected catalog render. The global GSVs are overridden by the catalog render marked as the **Front** buffer item, indicated by the icon.

To override the global GSVs with a catalog render:

1. Left-click the catalog render you want to work with to set it as the Front buffer.

   The catalog render is marked with the icon.

2. Enable the **Sync to Project Settings** checkbox in the **Catalog** tab menubar.



The global GSVs in the **Project Settings** are overridden by those selected for the catalog render marked with the Front buffer icon.

3. Toggling the **Monitor Layer** button off and on in the Viewer now allows you to confirm that there are no mismatches between your live render and scene graph geometry.



A live render overlaying scene geometry with the **Monitor Layer** enabled



The scene geometry with the **Monitor Layer** disabled for comparison

Foresight+ makes multi-shot workflows even quicker and more efficient inside Katana by improving the iterative process. You no longer need to work on one single shot, without knowing if their lighting looks consistent in the other shots. Katana makes it easy to work on an entire sequence inside one Katana project while making per-shot overrides, and seeing every change you make live, inside Katana.

Katana Queue

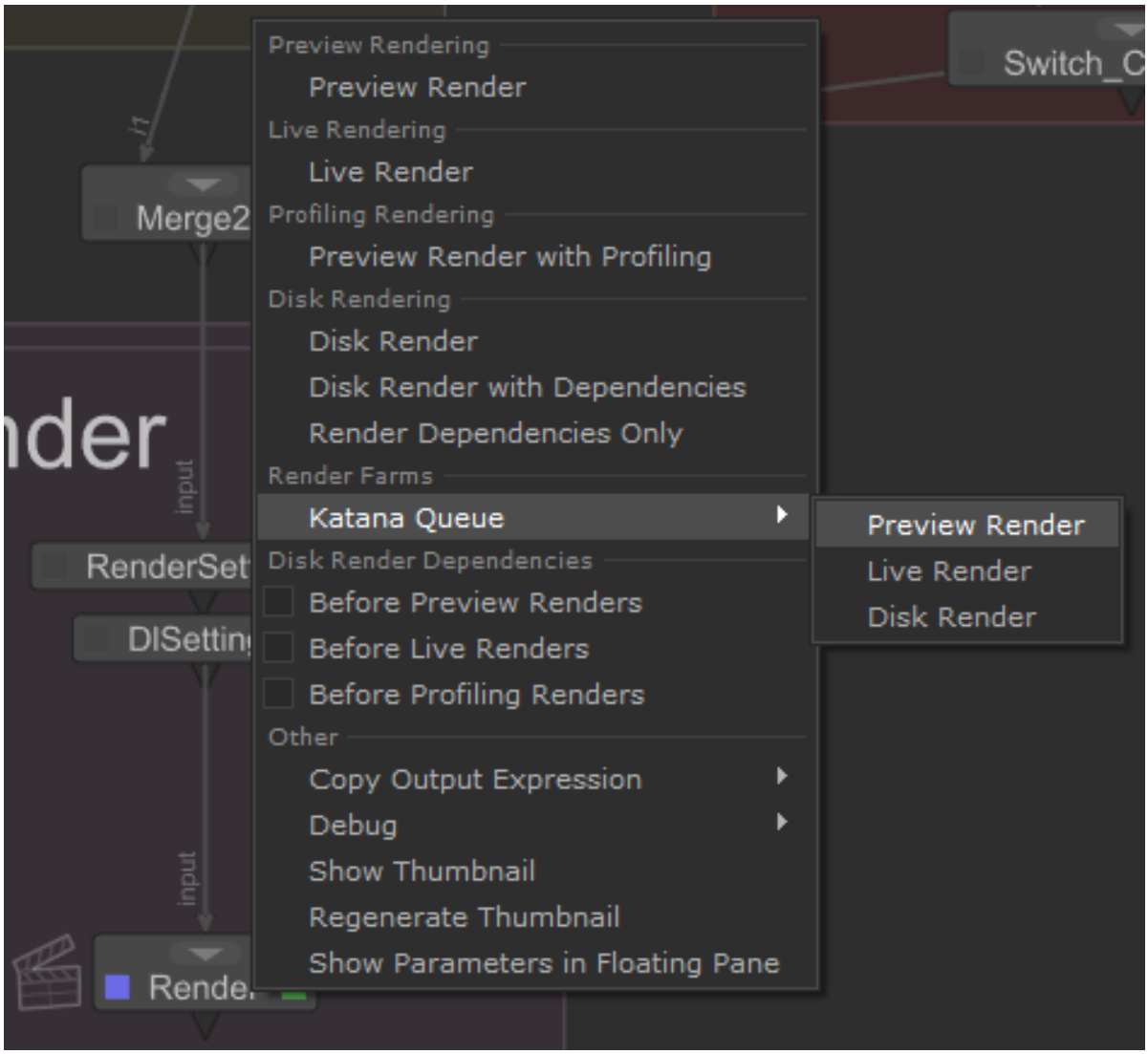
Katana Queue is a minimal render farm implementation, integrated with Katana using a custom render farm plug-in. The Katana Queue system can be used for the management of multiple renders across your local machine, or a set of machines on the same network, boosting rendering capabilities and increasing productivity.



Note: For more information, refer to the FarmAPI documentation in the Katana Developer Guide.

To start a render using Katana Queue:

1. Right-click the 3D node that you would like to start your render from.
2. Hover over the **Katana Queue** option and choose the type of render you would like to start.



The render starts and can be viewed in the **Monitor** tab, the **Catalog** tab and the **Monitor Layer**.

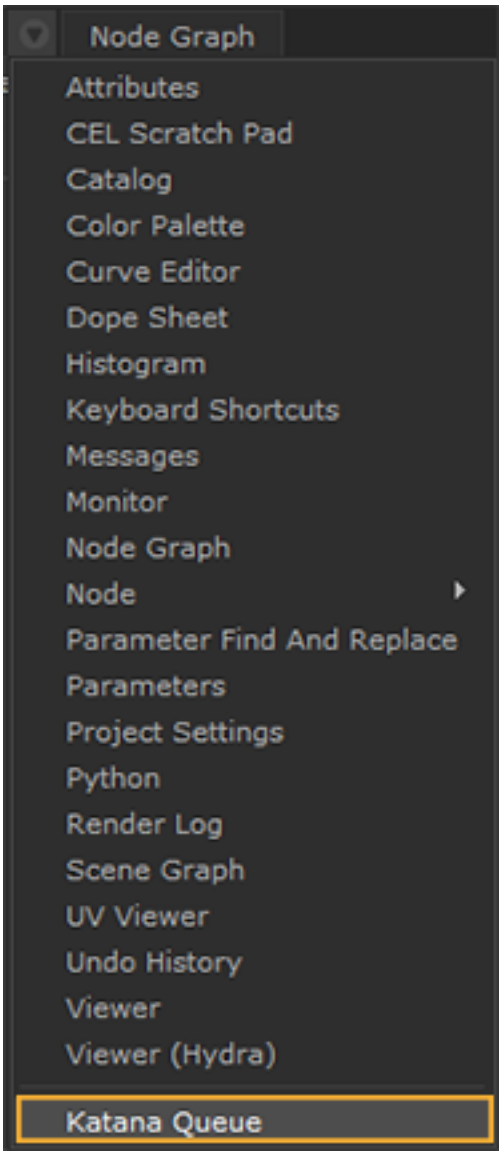
Render jobs that are running through the Katana Queue system can be viewed in the **Katana Queue** tab.

The Katana Queue Tab

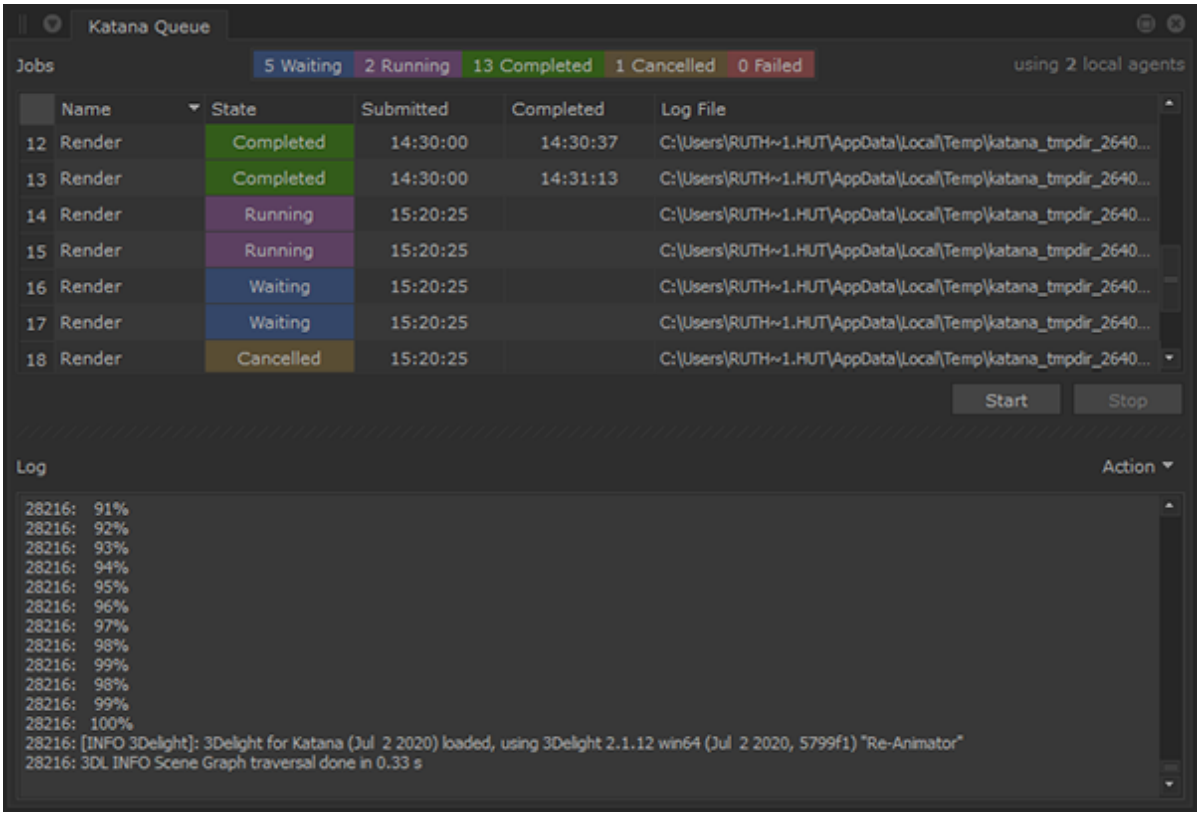
The **Katana Queue** tab provides a process control interface where you can view each render job which has been run using the Katana Queue system.

To access the **Katana Queue** tab:

1. Click the **Add Tab** button in the pane to which you would like to add a **Katana Queue** tab.
2. Click **Katana Queue**.



The **Katana Queue** tab opens.

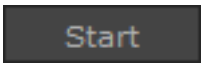


The **Katana Queue** tab is divided into two sections:

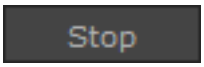
- **Jobs** - A list of render jobs that are being processed using Katana Queue.
- **Log** - A render log of the selected **Job**.

The **Jobs** section provides the following information about each render job:

- **Name** - The name of the node from where the render was started.
- **State** - The status of the render. The possible states are: **Waiting**, **Running**, **Completed**, **Cancelled**, **Failed**.
- **Submitted** - The time that the render was submitted to the Katana Queue.
- **Completed** - The time that the render was completed. If the render did not finish due to failing or being canceled, this field remains empty.
- **Log File** - The file path of the render log file.

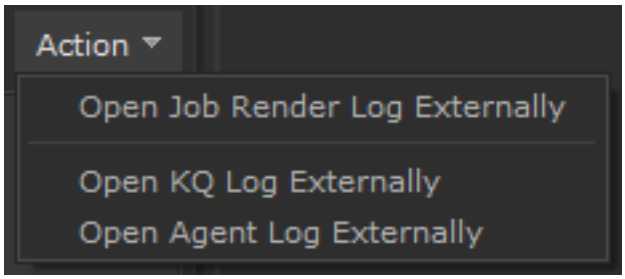
The **Start** button 

can be used to restart any renders from the **Katana Queue** render jobs list that had previously failed or been canceled.

The **Stop** button 

can be used to cancel any renders from the **Katana Queue** render jobs list that are currently running or waiting to start.

The **Log** section provides all render log information, and an **Action** button, from which you can open your **Job Render Log**, **KQ Log** or **Agent Log** externally.



Note: The external text editor application can be specified in the Katana **Preferences**. Open the **Preferences** dialog by choosing **Edit** > **Preferences** from the main menu, and navigate to **externalTools** > **editor** to select your text editor.

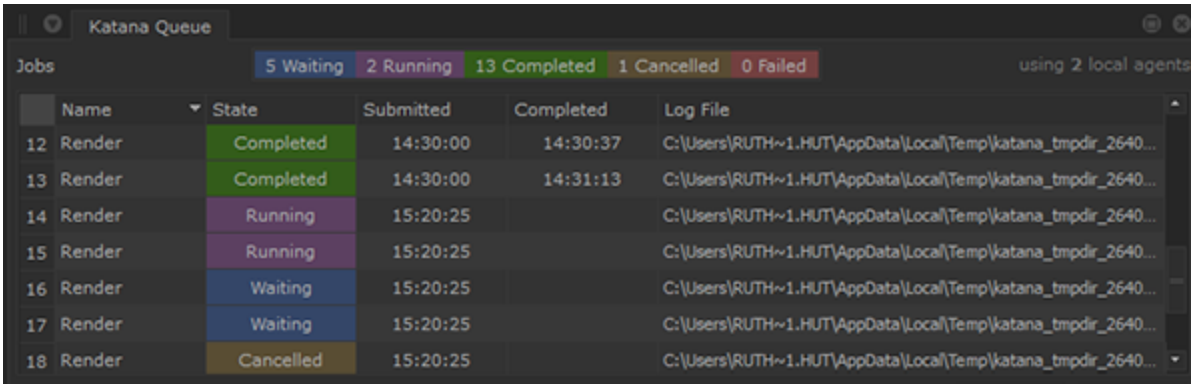
Working with Katana Queue Jobs

The render jobs can be filtered and sorted in the **Katana Queue** tab the following ways:

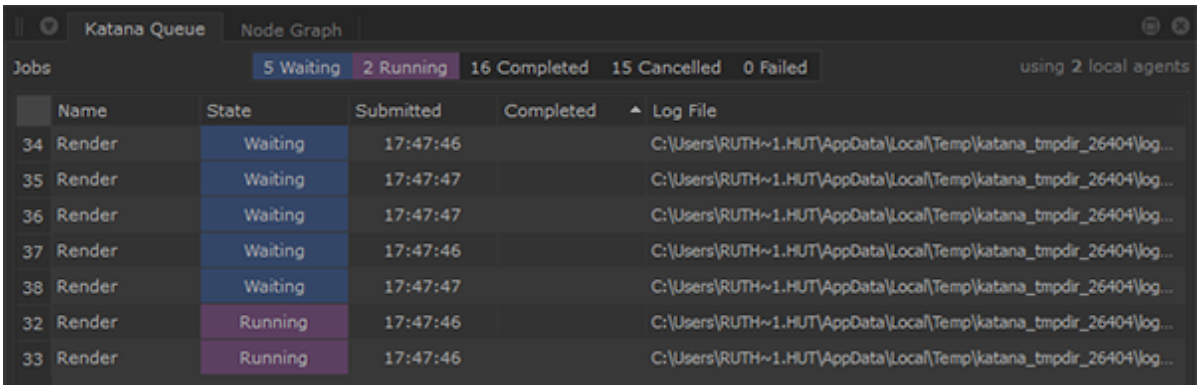
- Click on the filters at the top of the **Katana Queue** tab to choose which states you want to be visible in the render jobs list.



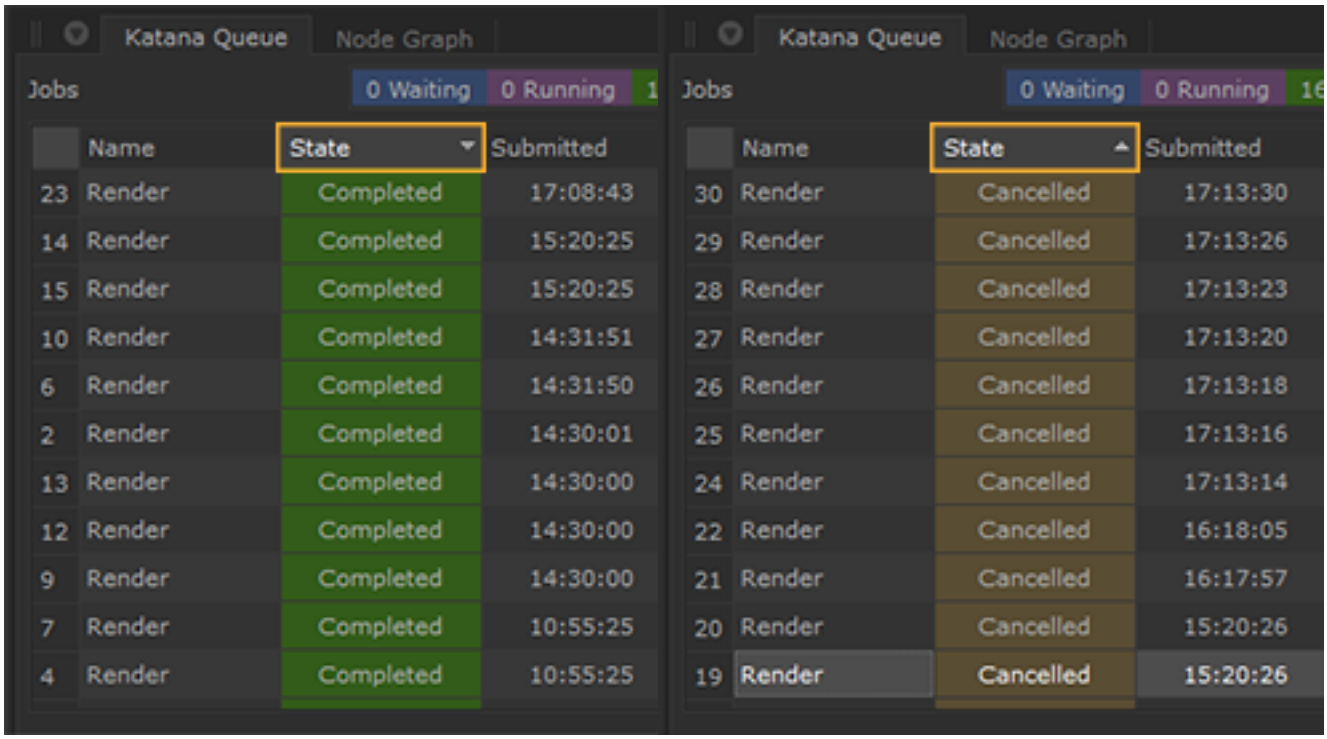
When all filters are turned on, all render jobs are visible in the list.



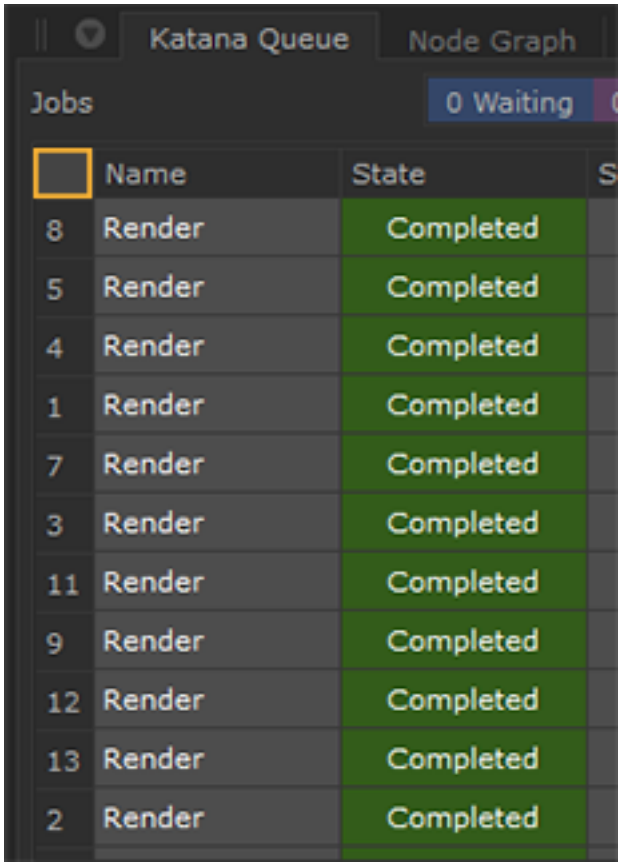
Filters can be applied to show only the render jobs you are interested in seeing.



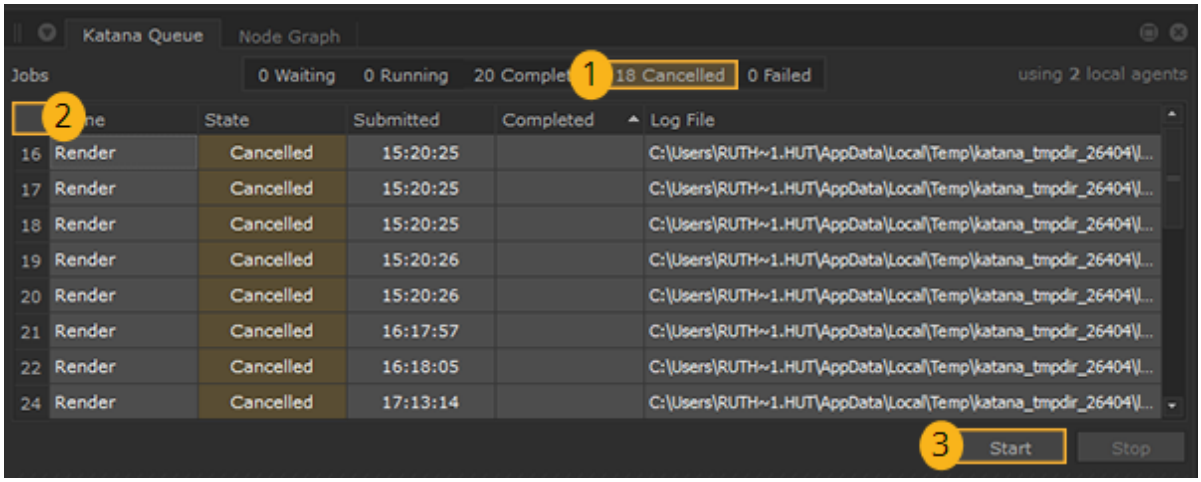
- Click on a column name, such as **State** or **Submitted**, to order the render jobs according to the selected information. Click again to toggle whether the jobs are arranged in ascending or descending order.



- Click the empty square in the top left of the **Jobs** list to select all jobs in the list.



This can be useful if you want to restart all **Cancelled** renders. You can use the filters to show only the canceled jobs then select all and click the **Start** button.



Configuring a Render

These pages explain how rendering can be configured in Katana to suit your requirements.

Render Dependencies

How to find which nodes the Render node depends on.

Rendering only Selected Locations

How to render only selected objects.

Setting up Interactive Render Filters

How to easily set up common recipe changes for interactive and live renders.

Managing Color

Using the OpenColorIO standard to manage color data.

Render Dependencies

The function **GetSortedDependencyList()** provides the information needed to obtain the names of any other nodes a Render node depends on. For example, the following script returns the names of a Render node's dependencies, as a sequence:

```
def dependencyList(nodeName):
"""
Use GetSortedDependencyList to retrieve the entire dependency tree for a
render node.
Each entry is ordered so that render nodes are sorted by number of
dependencies, in descending order.
"""
# Get hold of the node and all of its dependencies as a sequence of
dictionaries
node = NodegraphAPI.GetNode(nodeName)
info = FarmAPI.GetSortedDependencyList( [ node ] )
# Extract the 'deps' for each entry in the sequence
```

```
# to produce a flat list.
allDeps = [ dep for i in info for dep in i["deps"] ]
return allDeps
```

Rendering only Selected Locations

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.

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, or the number of light bounces. A filter can consist of more than one change to the recipe and it is the equivalent of appending the filter nodes to the end of the node you selected to render.

These filters are bundled together inside the InteractiveRenderFilters node and toggled using  at the top of the interface.

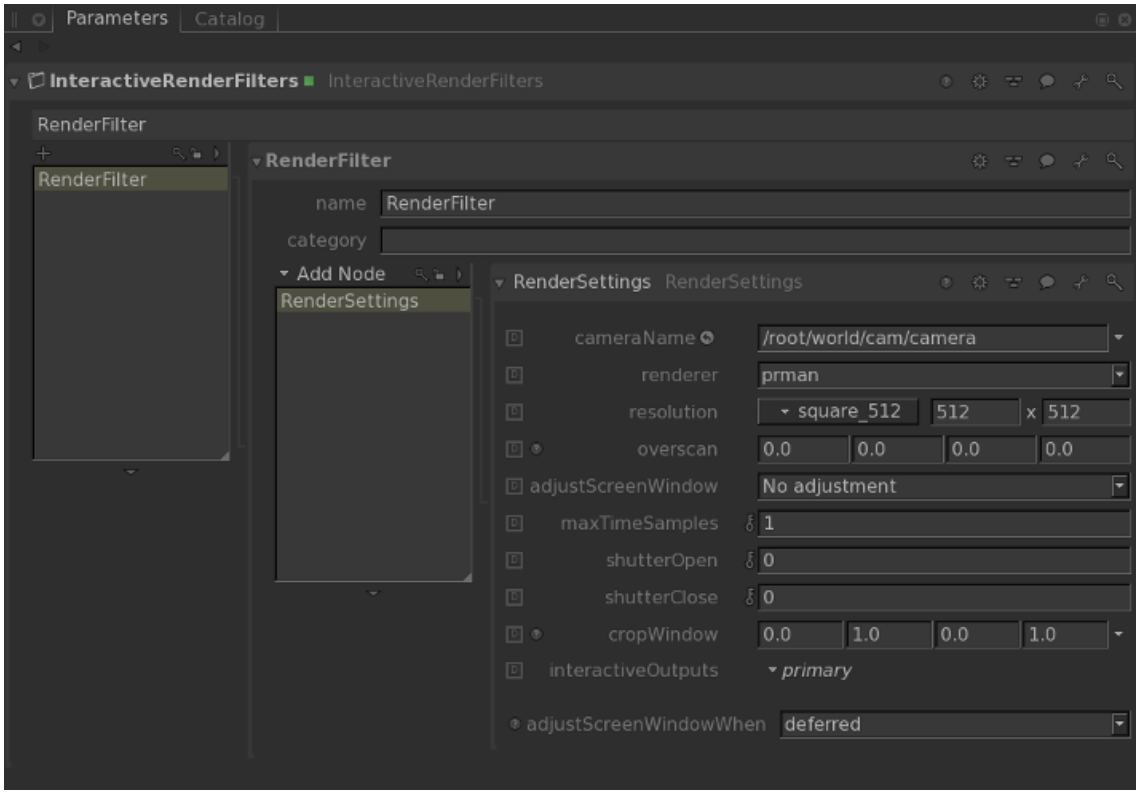
Creating Interactive Render Filters

1. Create an InteractiveRenderFilters node and place it anywhere within the **Node Graph**.
2. Select the InteractiveRenderFilters node and press **Alt**+**E**.
   The InteractiveRenderFilters node becomes editable within the **Parameters** tab.
3. Click  below **RenderFilters** in the **Parameters** tab.

A new **RenderFilter** is created.

4. To help you remember what this filter does, type a name in the **name** parameter.
5. To create a group of filters, type a group name in the **category** parameter.
6. Click **Add Node** and select a node from the list.

   Filter the list using the **Categories** dropdown or the **Filter** field.
7. Make any changes to the node.



8. Repeat steps 6 and 7 for any additional nodes for this filter.
9. Repeat steps 3 to 7 for any additional filters.

Note: The InteractiveRenderFilters node doesn't need to be connected into a recipe to work.

Tip: It is also possible to middle-click and drag nodes from the **Node Graph** tab into the **Add Node** list of the InteractiveRenderFilters node.

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.)

Managing Color

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_lnf.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, the **Viewer**, or the **Catalog** tab. The **Monitor** tab is a viewer for current and previous renders. The **Viewer** features a **Monitor Layer**  which can be toggled to view your renders over the top of the geometry in the viewer. 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

Learn to view your renders using Katana's Monitor tab.

Using the Catalog Tab

Use the Catalog tab to organize and view your renders.

Using the Histogram

Find out how to check RGBA levels within an image using the Histogram tab.

The Monitor Layer

Learn to view your renders using Katana's Monitor Layer in the Viewer.



Article: If you are experiencing issues when viewing renders in Katana, please refer to the Knowledge Base article: Troubleshooting Rendering Issues in Katana

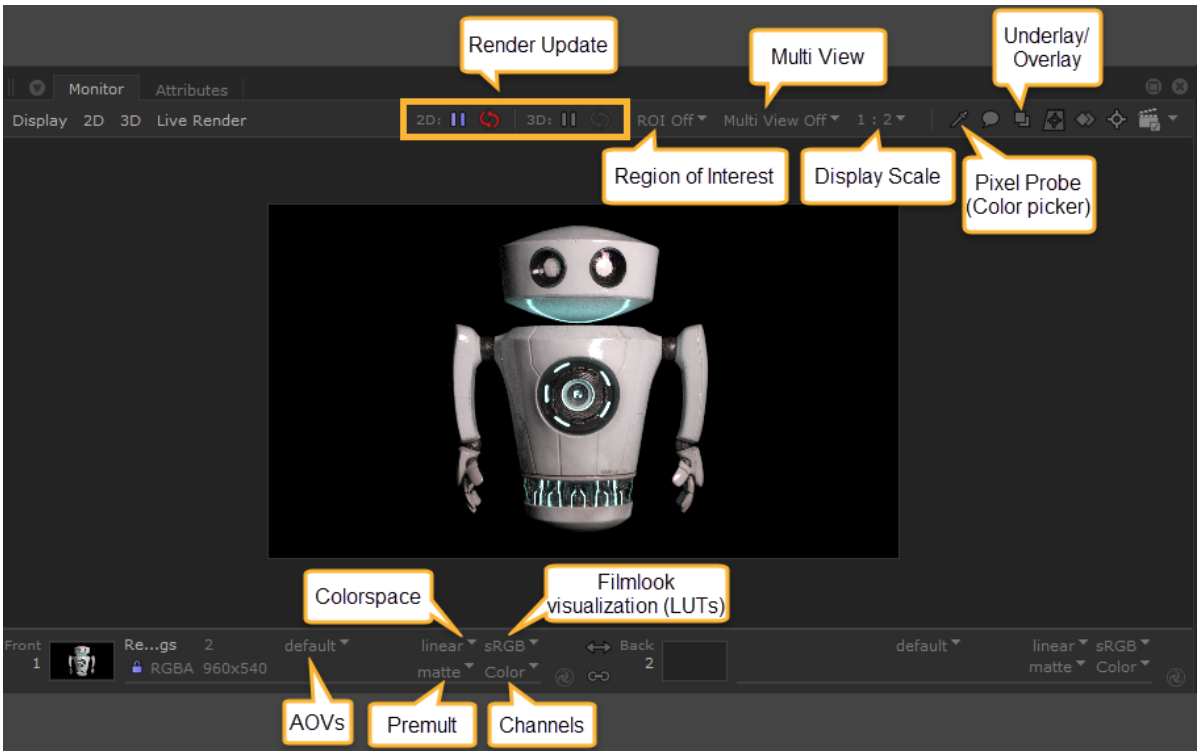
Using the Monitor Tab

The Monitor tab can be used to view your renders inside Katana.

Monitor Tab

Once a render has completed, you will be able to see the image in the **Monitor** tab.

Monitor Tab Controls



An overview of the key controls to edit how your image is displayed:

- **Render Update** controls - Choose a 2D or 3D update mode to determine when to trigger a render. See more at Changing How to Trigger a Render.
- **Region of Interest** dropdown - Use this to set a region of interest to render. For more information, see Rendering a Region of Interest (ROI).
- **Multi View** dropdown - If you want to view front and back at once, you can change the layout, including A/B buffer multiview and swipe modes. See more at Comparing Front and Back Images.
- **Display Scale** dropdown - If you want your rendered image to fill the whole Monitor tab window, you can use Frame Display Window or Frame Data Window, or you can choose a custom zoom. See more about changing the image position at Changing the Image Size and Position.
- **Pixel Probe** (Color picker) - This tool helps if you want to identify geometry by selecting pixels from the Monitor tab. Opens a new toolbar where you can select Front and back pixels. See more at Viewing the Pixel Values.

- **Underlay/Overlay** controls - Opens a new toolbar in which you can underlay or overlay the image with either the Over or Add function. See more at Underlaying and Overlaying an Image.
- **AOVs** dropdown - Use this dropdown to select which AOV to display in the Monitor tab. See Selecting An Output Pass.
- **Premult** dropdown - Change the alpha channel and premult settings. See more at Changing How the Alpha Channel is Displayed.
- **Channels** dropdown - Select the channel to display. See more at Changing the Displayed Channels.
- **Colorspace** dropdown - Select which colorspace to display. The default is 'linear'.
- **Filmlook visualization (LUTs)** dropdown - Apply different Look Up Tables (LUTs) to the image. To add more options to this list, you can append to the "displays" section of the config.ocio file. Learn more about this at Q100197: Colorspaces in the Monitor Tab.

Tip: There are many useful keyboard shortcuts when using the Monitor tab. See Monitor Shortcuts.



Warning: In the Monitor Tab, **Ctrl**+**Up Arrow/Down Arrow** is used to increment or decrement the display **exposure** (fstop) offset by 1/4 stop. **Ctrl**+**Home** resets display exposure (fstop) offset back to 0.



Tip: **Alt**+**Up Arrow/Down Arrow** adjusts the **gamma** of the image.



Note: The Monitor tab supports the application of custom overlay masks with different framing options, based on a supplied configuration file. See more at Overlay Masking.

Viewing 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.

To toggle 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**.

To switch the front and back images, hold the mouse pointer inside the image display area of the **Monitor** tab and press **S**.

To change 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.

To view the catalog from inside the **Monitor** tab, press the **Tab** key. (Pressing the **Tab** key again returns to the **Monitor** view.)

Monitor Layer

The Monitor Layer allows you to toggle a render view directly in the Hydra Viewer, overlaying the geometry.

For a full overview of the Monitor Layer, see The Monitor Layer in the Hydra Viewer.

Changing the Image Size and Position

There are numerous ways to get the image to the right size and location within the **Monitor** tab.

To move the image around the **Monitor** tab, middle-click and drag.

To fit an image to the **Monitor** tab, at the top of the tab, select **[Current display ratio]** (for instance **1.23 : 1**) > **Frame Display Window** (or press **F**).

To 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, and the change is reflected in the display scale at the top of the tab, or
- Press **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 outlineColor. 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**, or
- 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 Rendering Your Scene.

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.

Rendering a Region of Interest (ROI)

To reduce the 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. You can turn on Region of Interest rendering from the **Monitor** tab and the **Monitor Layer** in the **Hydra Viewer**.

If the ROI is enabled in either one of the **Monitor** or the **Hydra Viewer**, it is automatically enabled in both.

ROI in the Monitor Tab

You can turn on Region of Interest rendering in the **Monitor** by selecting **[ROI menu]** > **ROI On** or **ROI On (visible)**. If you then want to turn it off again, select **[ROI menu]** > **ROI Off** or **ROI Off (visible)**. You can also toggle the state of the ROI by pressing **Shift**+RMB. This toggles between **ROI Off** and **ROI On (visible)**. Alternatively, select an area for the region of interest by **Shift**+RMB and dragging the mouse to create a selection box.



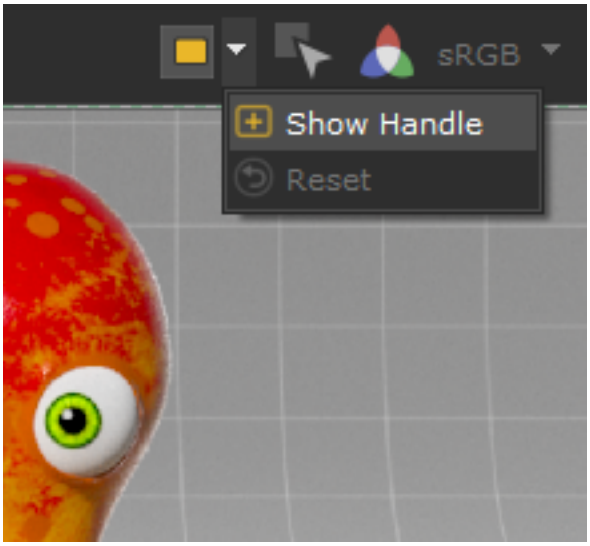
Note: If the region of interest's bounding rectangle is visible, you can change the bounds by dragging the edges.

Tip: Should you need to access the values of the ROI for scripting purposes, they are stored as parameters on the project root node.

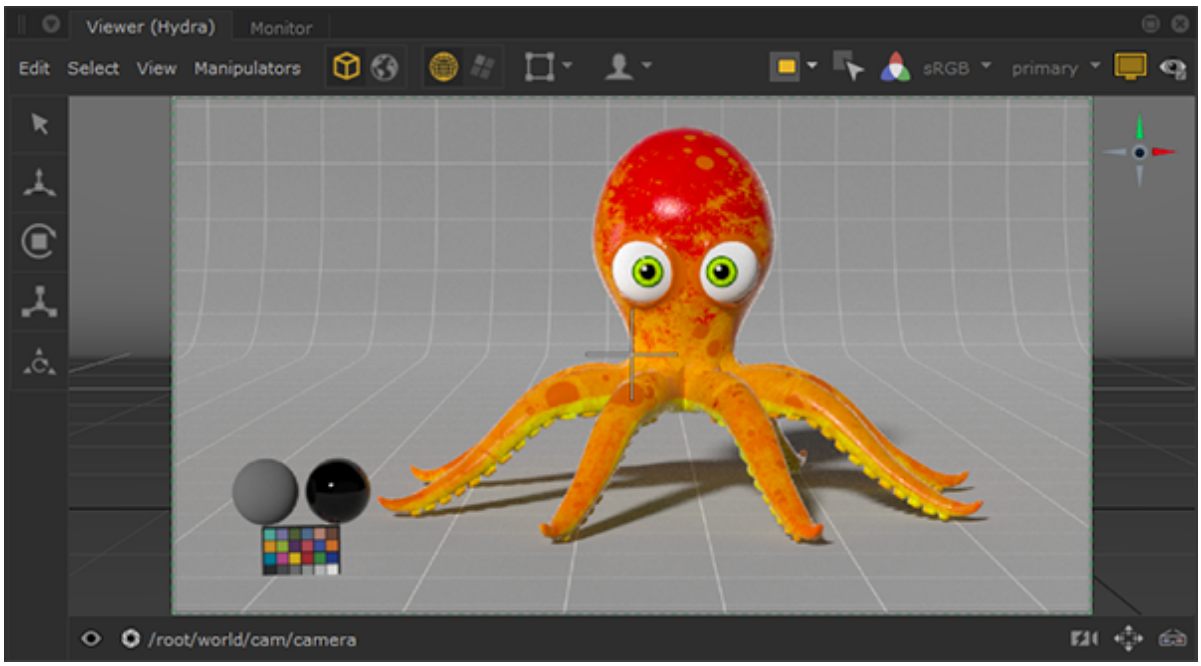
ROI in the Monitor Layer

To turn on Region of Interest rendering in the Monitor Layer:

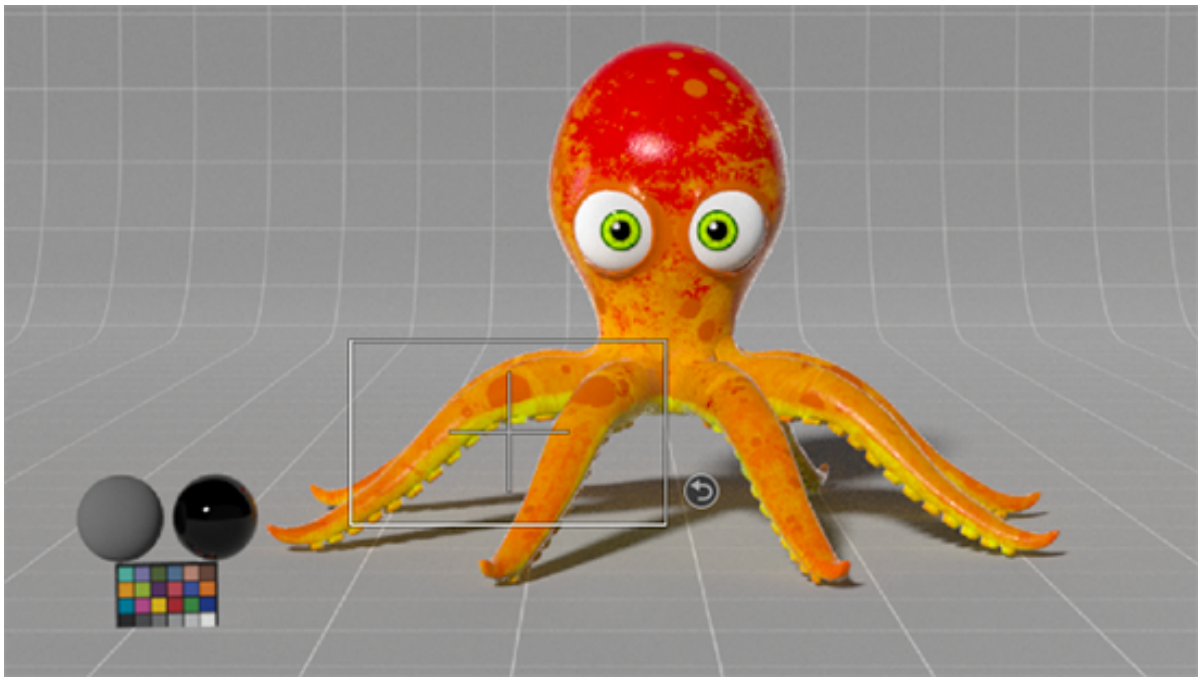
1. Click the **Region of Interest (ROI)** button  to enable Region of Interest rendering.
2. Click the drop-down arrow next to the **Region of Interest (ROI)** button and select **Show Handle**.



By default, the handle is set to the size of the camera screen window.

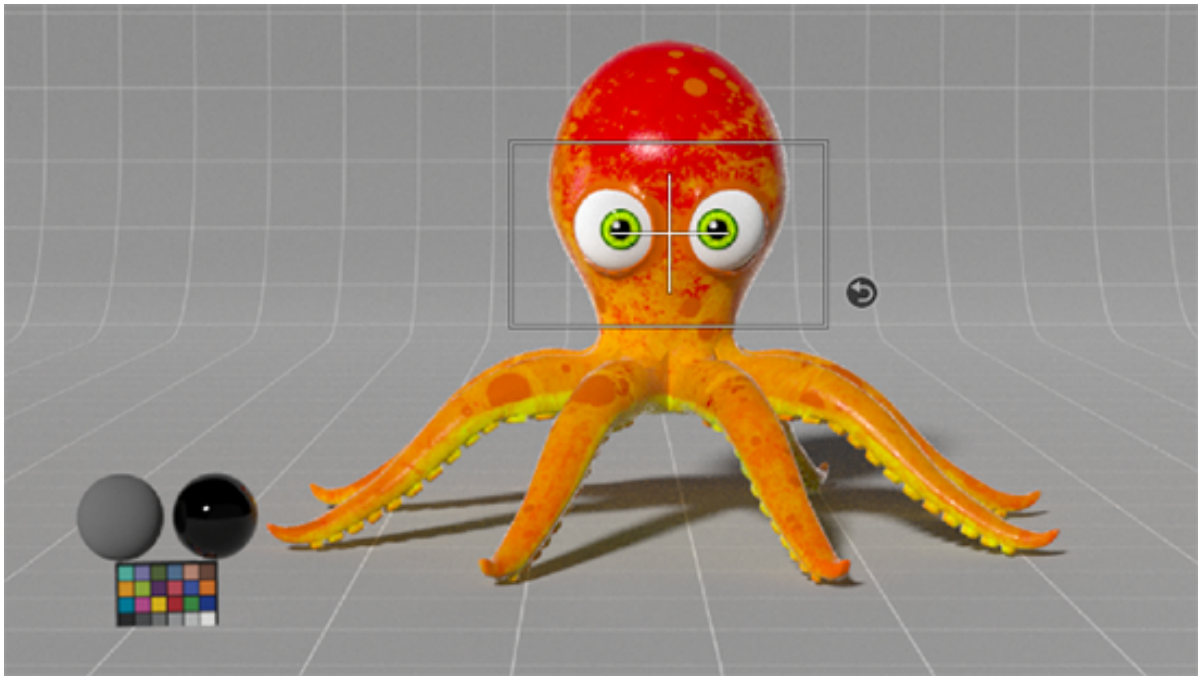


3. Click and drag an edge or corner of the handle to resize the ROI.

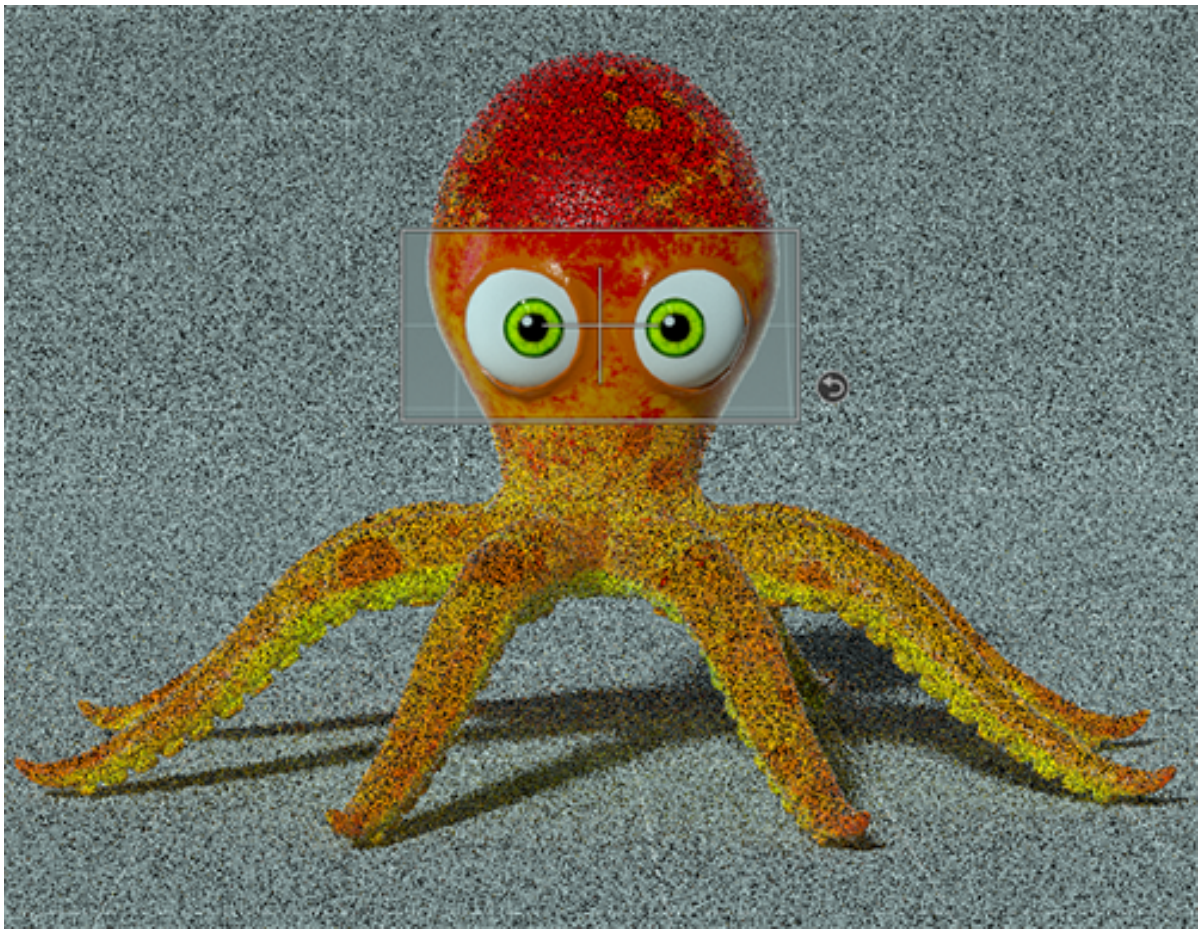


Note: You can also hold **Shift**, Right-Click and drag to draw the ROI.

4. Click and drag the center cross to move the handle.



5. Start a **Preview** or **Live Render.**

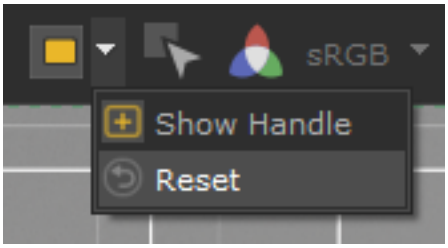


The Region of Interest is rendered.

> **Note:** Different renderers mange ROI rendering differently. Some may continue rendering the rest of the image, others may stop rendering once the ROI is complete.

To reset the Region of Interest handle, you can either:

- Click the **Reset** button next to the Region of Interest handle, or
- Click the drop-down arrow next to the Region of Interest (ROI) button and select **Reset**.



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.



To toggle premultiply in the Monitor tab, select **[Alpha display]** > **Premult (xA)**.

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:

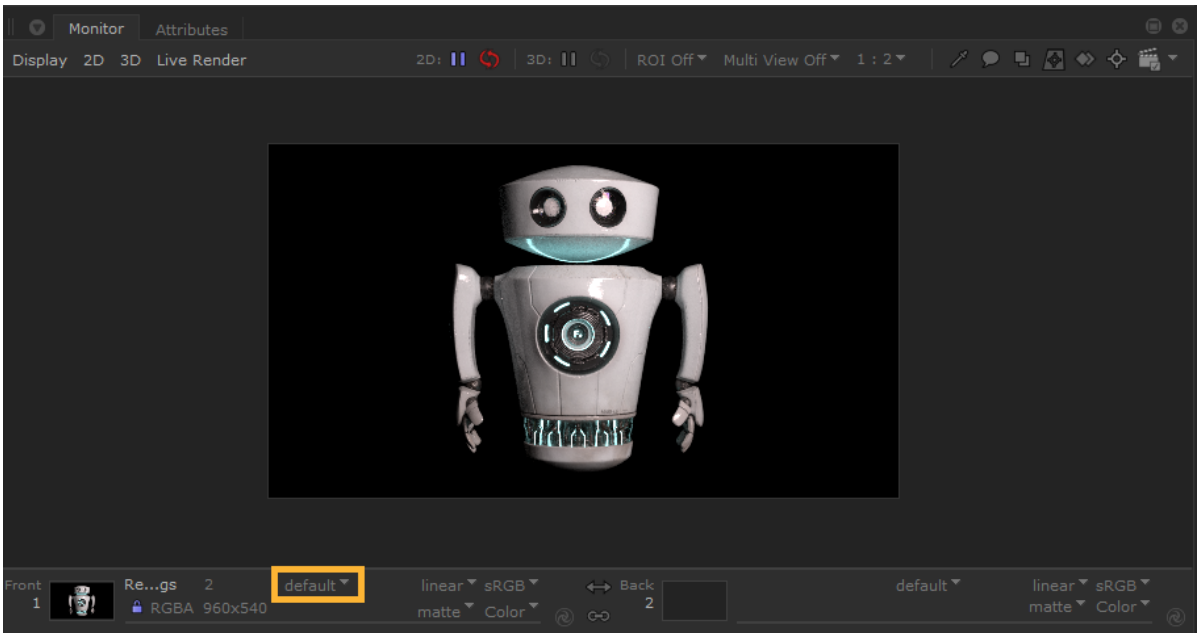
- **None** - No alpha overlay is displayed.
- **Mask** - The area of the image with no alpha channel becomes the overlay color.
- **Matte** - The area of the image with an alpha is overlayed with the overlay color.

To change the color used for alpha overlays:

1. Select **[Alpha display]** > **Set Overlay Color...** .
   The color picker dialog displays.
2. Select a color with the color picker.
3. Press **OK**.

Selecting An Output Pass

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).

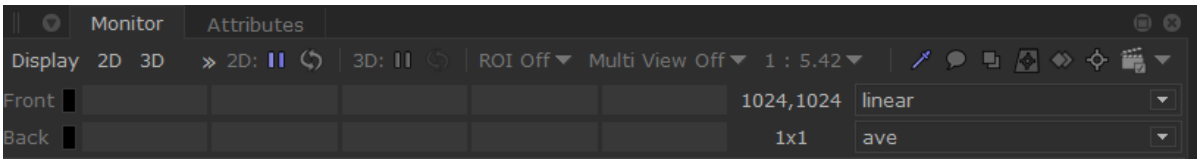


For details on setting up multiple outputs, see Defining an AOV Output. For more on sending those outputs to the **Monitor** tab, see Previewing Interactive Renders for Outputs Other than Primary.

Viewing the Pixel Values

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 by selecting  . The pixel probe toolbar displays.



2. To change the colorspace for the displayed pixel values, select from the top dropdown in the pixel probe toolbar.

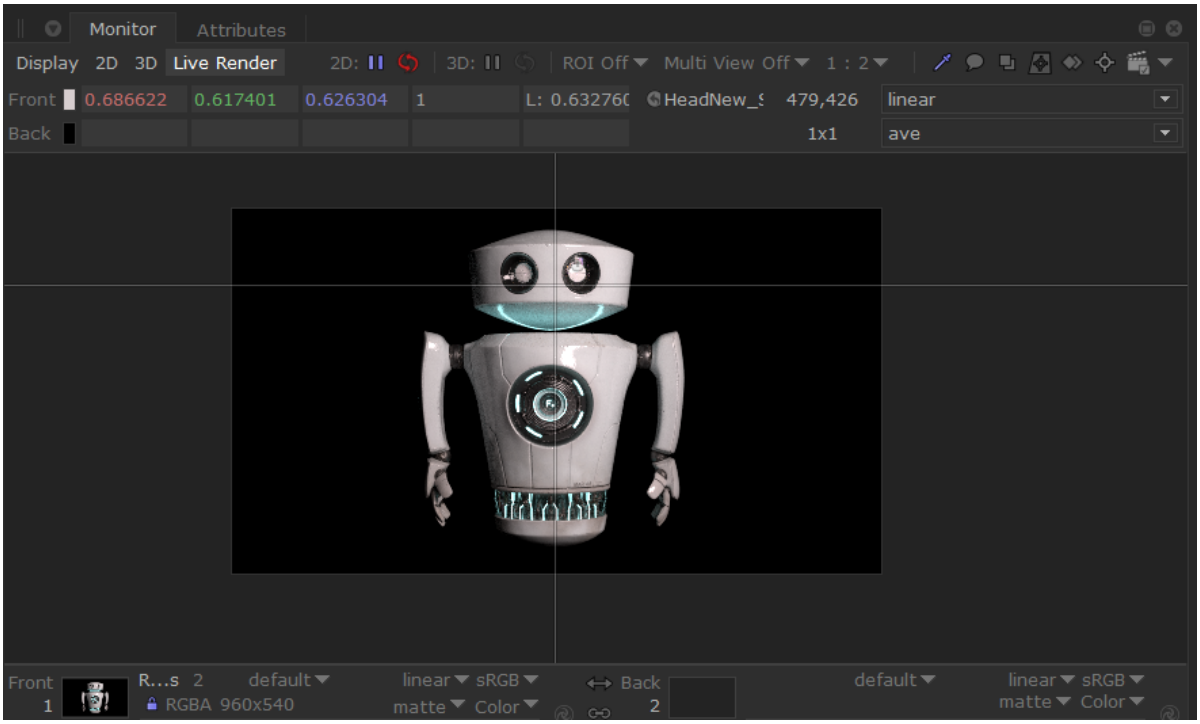
   To change what type of pixel value you want displayed, click the lower dropdown in the pixel probe toolbar and select:

   - **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.

3. Pick a pixel on the object in the Monitor.

   To change where the pixel probe samples:

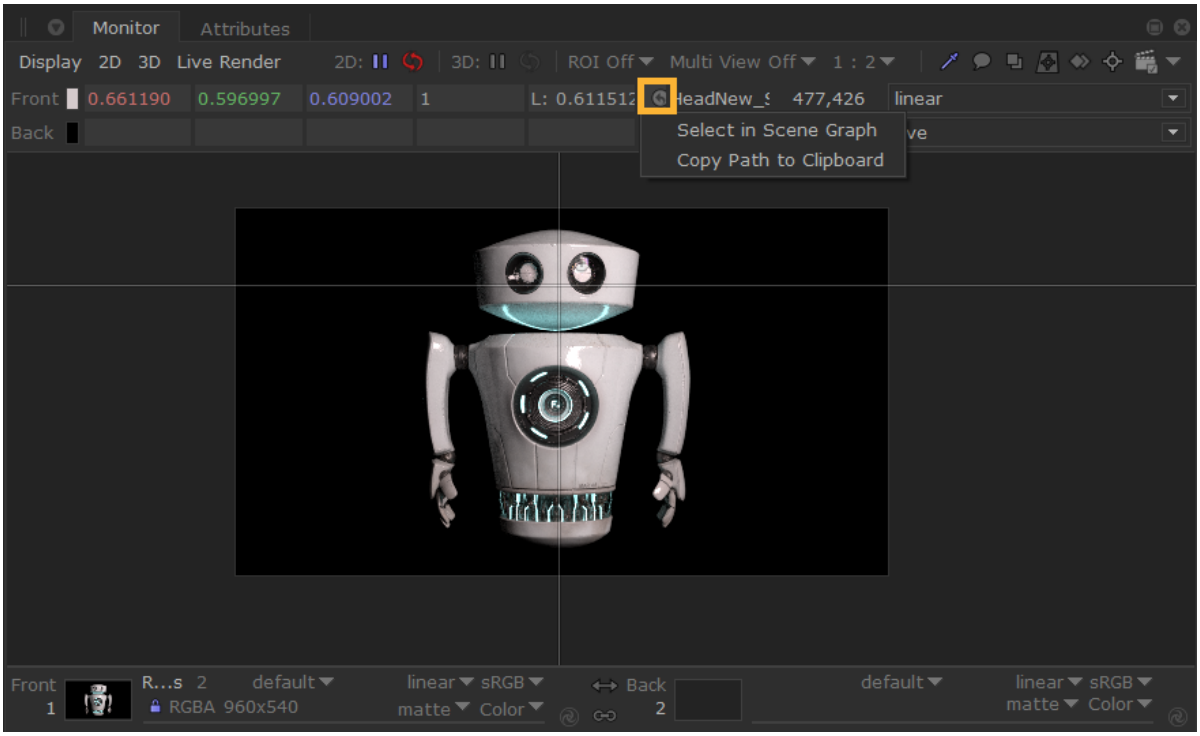
   - **Ctrl**+click to change the sample center point.



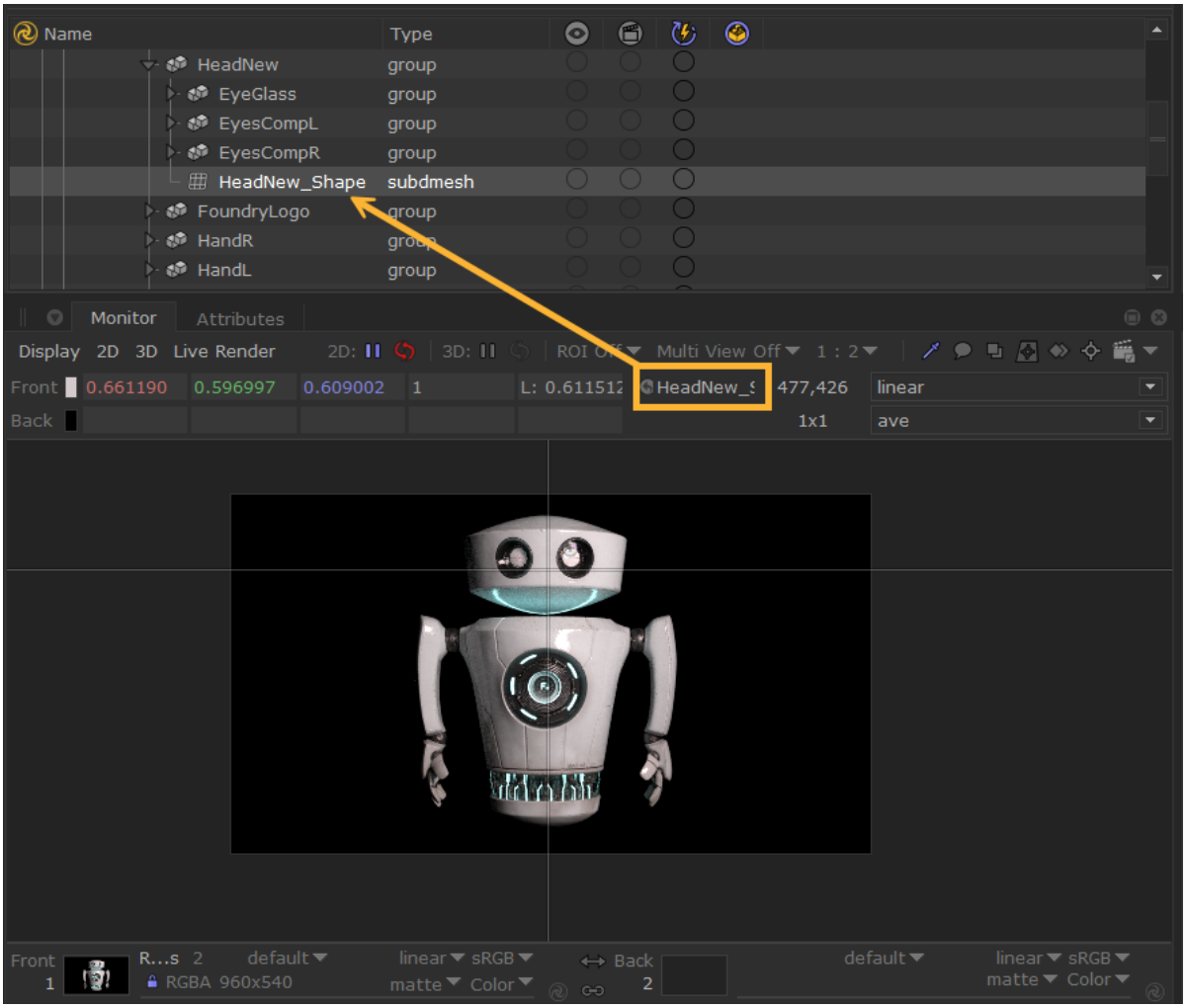
- Click and drag the center point.
- Click and drag the vertical bar to move the sample's center left and right.
- Click and drag the horizontal bar to move the sample's center up and down.
- When sampling an area, click and drag the bounding border lines to change the area's bounding rectangle.

Katana's internal 'id' channel mapping identifies the geometry that this color originates from.

4. Click on the scene graph locations widget that appears next to the geometry name in the pixel probe toolbar, and click **Select in Scene Graph** from the dropdown menu.



The scene graph location is highlighted in the **Scene Graph** tab.



5. To turn off the pixel probe, click  or press **.** (period). The pixel probe toolbar is hidden.

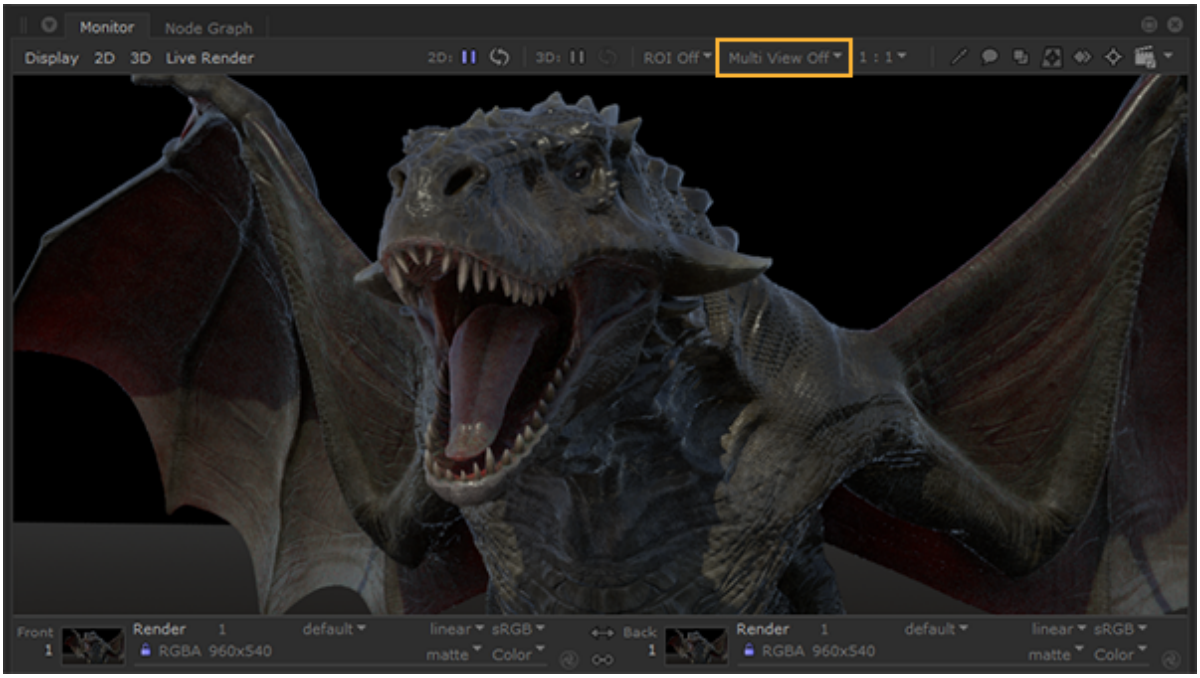


Note: For more information on the **Monitor** tab, refer to Using the Monitor Tab.

Comparing Front and Back Images

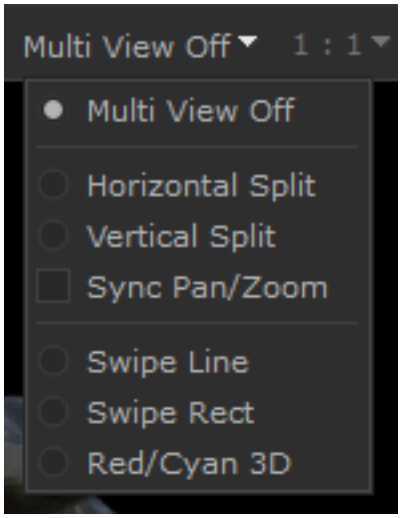
If you need to compare the **Front** and **Back** images from the **Catalog** tab, for instance, to see how changes are affecting an image or to ensure that colors are consistent across shots, you can use the **Multi View** feature within the **Monitor** tab.

Note: For more information about Front and Back images, see the Changing the Catalog Renders Displayed in the Monitor Tab section of the **Catalog** tab topic.

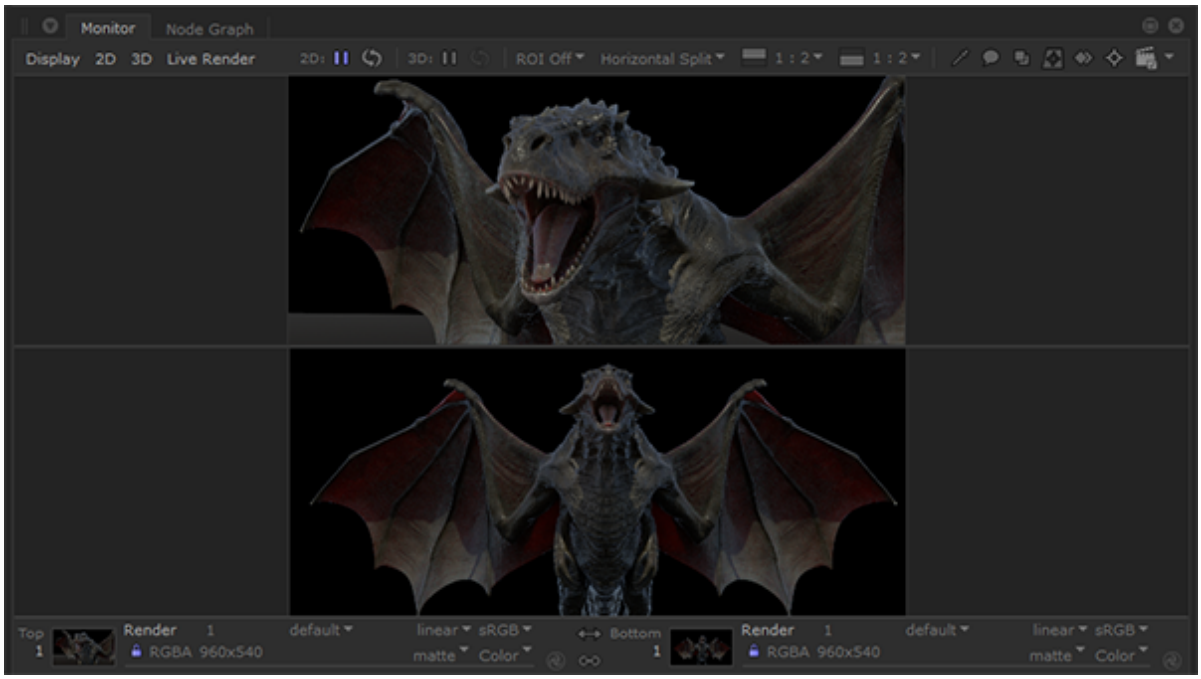


Multi View Options

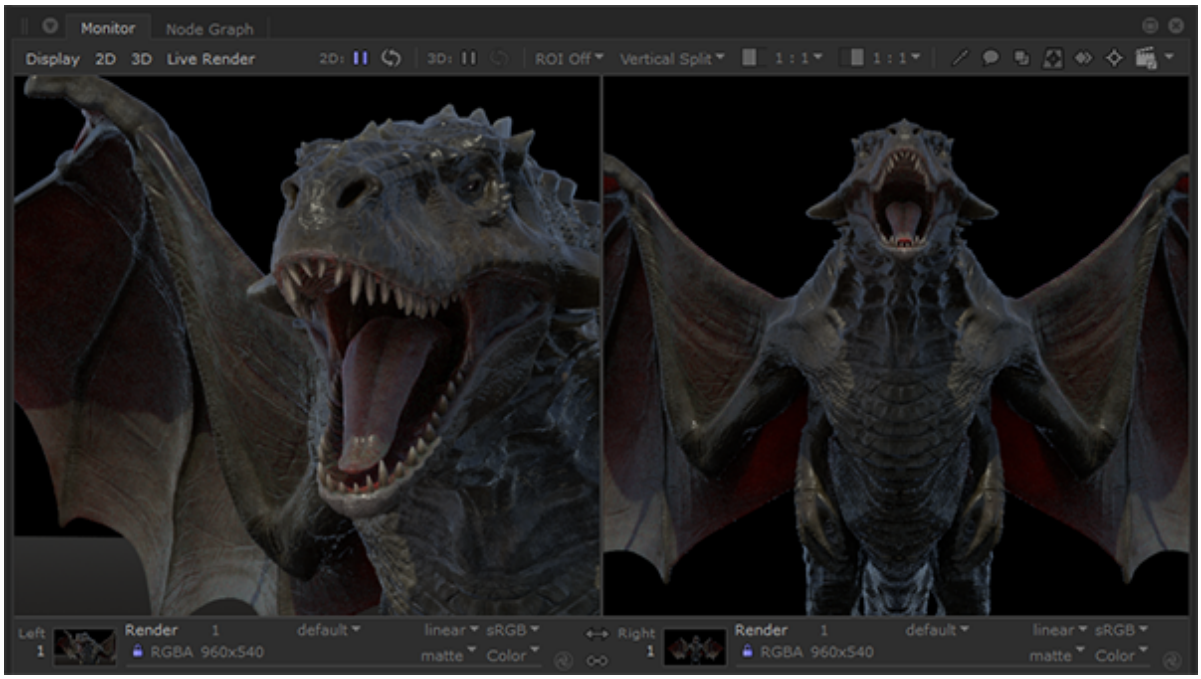
Click the **Multi View Off** drop-down to select a **Multi View** mode.



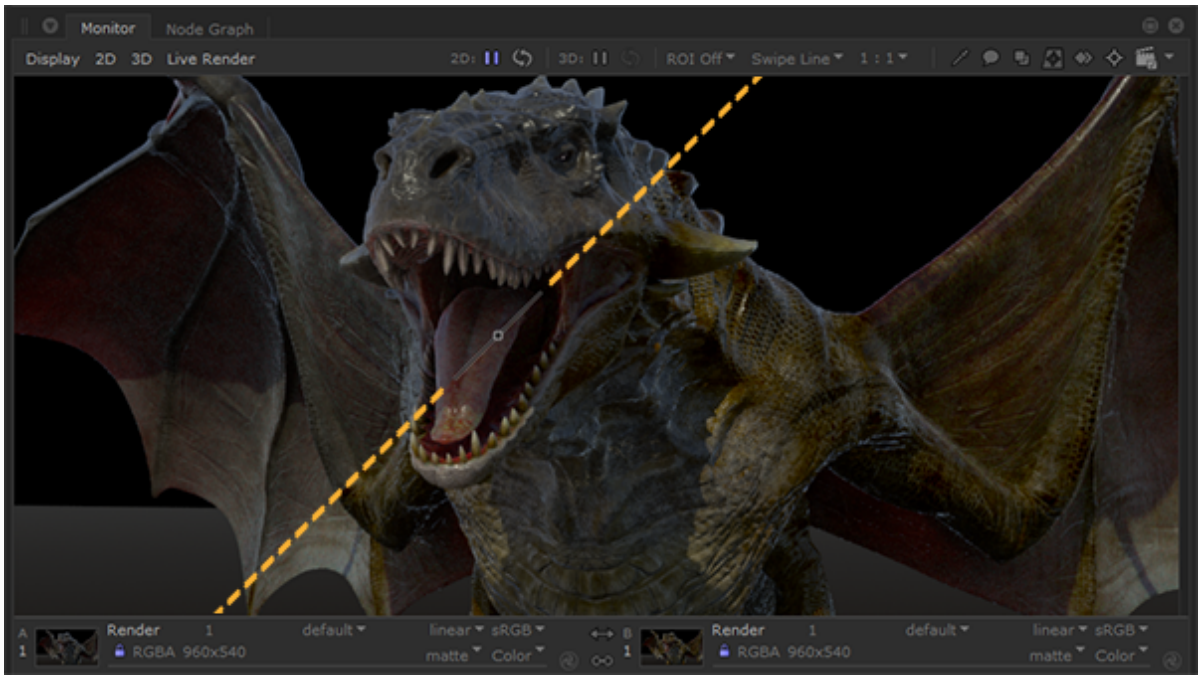
- **Horizontal Split** - Divides the **Monitor** tab horizontally into two sections. The **Front** image is displayed at the top and the **Back** image is displayed at the bottom.



- **Vertical Split** - Divides the **Monitor** tab vertically into two sections. The **Front** image is displayed on the left and the **Back** image is displayed on the right.

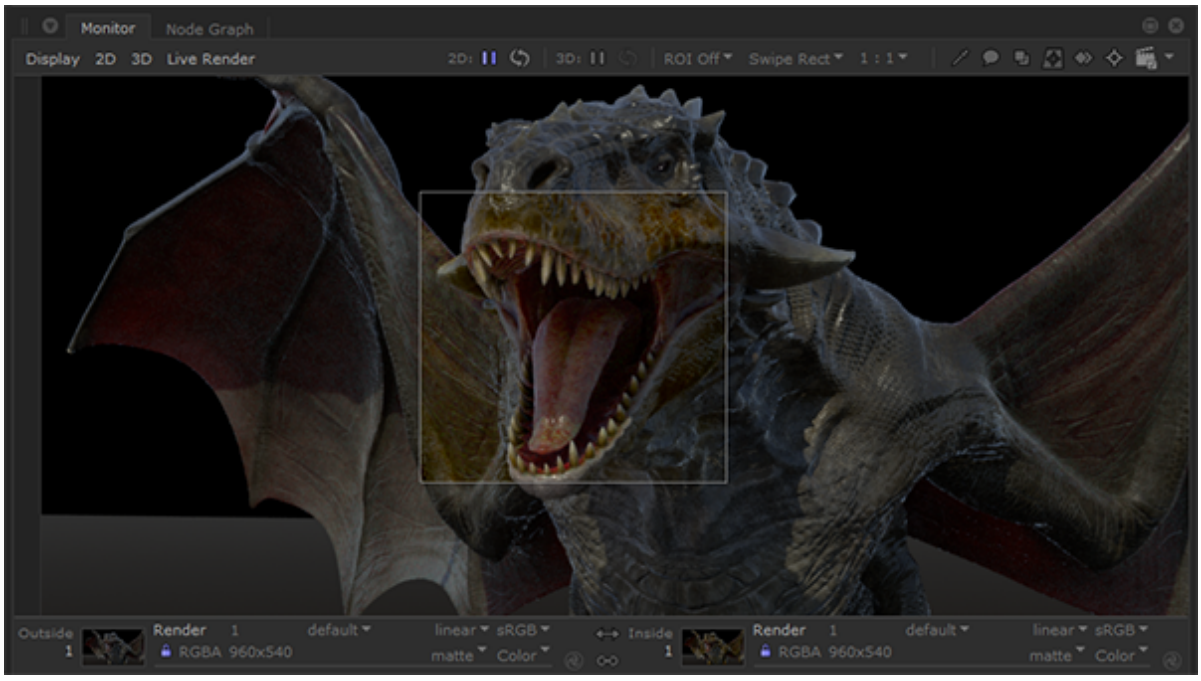


- **Sync Pan/Zoom** - If this option is enabled when using **Horizontal Split** mode or **Vertical Split** mode, panning and zooming on any one image is synchronized to both images.
- **Swipe Line** - A line handle acts as a curtain from one image to the next. Click and drag the center of the handle to move its origin. Click and drag the lines either side of the handle's center to change the swipe angle.



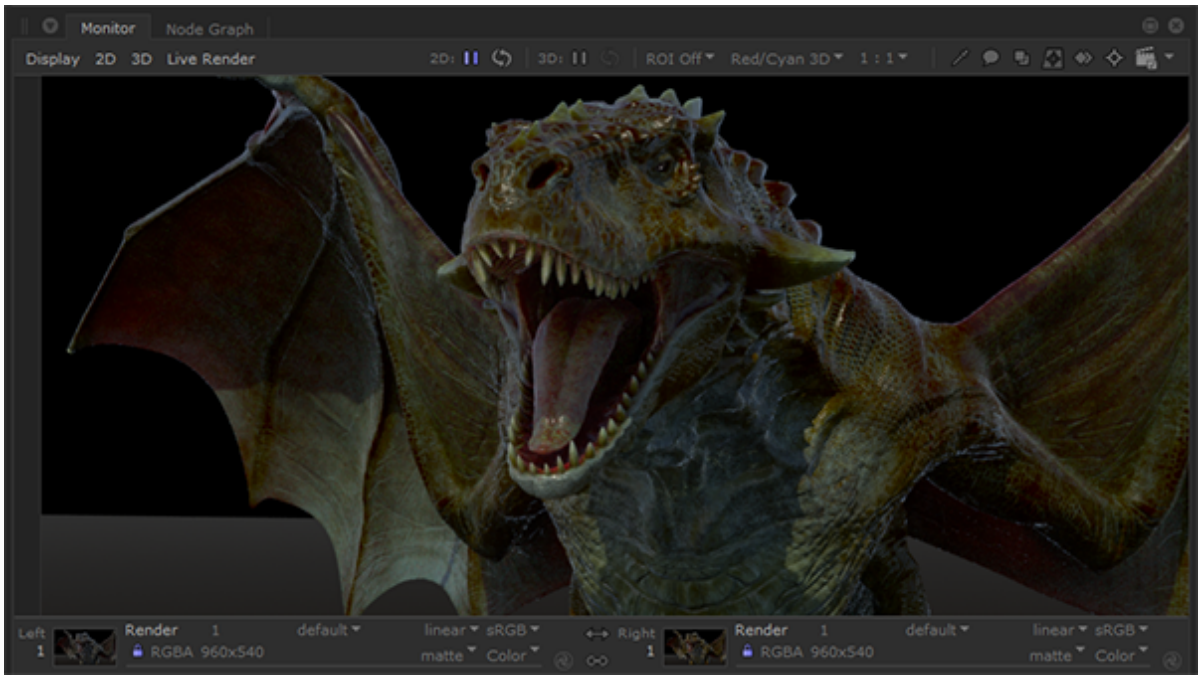
The names for the **Front** and **Back** images become **A** and **B** respectively.

- **Swipe Rect** - A bounding rectangle displays the **Back** image inside the rectangle and **Front** image outside the rectangle. Click and drag the center of the handle to move its origin. Click and drag the bounding box lines to resize the swipe rectangle.



The names for the **Front** and **Back** images become **Outside** and **Inside** respectively.

- **Red/Cyan 3D** - A mix between the **Front** and **Back** images.



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. In order to display the underlay and overlay controls, click . The **Underlay**/**Overlay** toolbar is added to the **Monitor** tab.

If you want to add 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. To remove an image from the underlay or overlay fields, click ✖ to the right of the image.

To turn off the underlay or overlay composition, uncheck the checkbox to the left of the field name. Whereas if you want to change 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**.

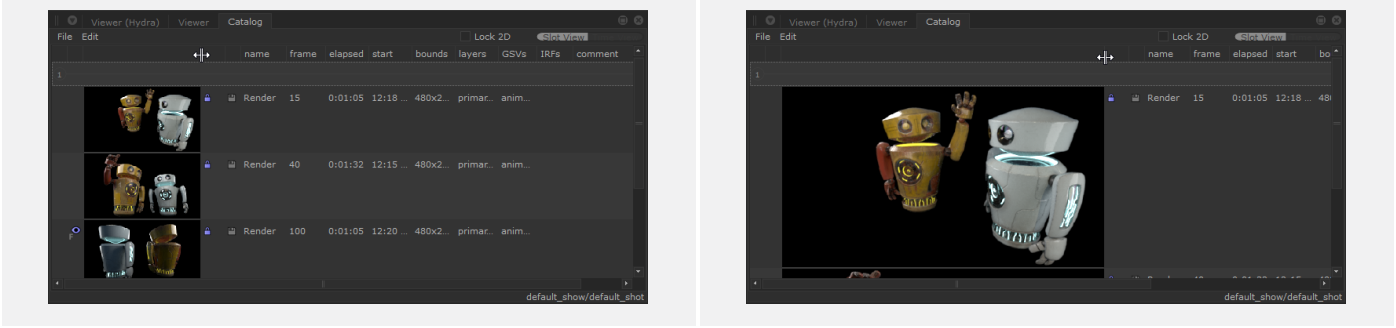
If you want to remove the underlay/overlay toolbar altogether, click .

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.

Note: For more on changing which slot to use, see Using the Monitor Tab.

The **Catalog** tab allows you to view and organize your renders. The **Catalog** tab also displays render settings, such as the frame, render time, Graph State Variables or Interactive Render Filters. The thumbnails display both complete and running renders, and can be resized by scaling the column size.



Resizing thumbnails in the Catalog tab

Customizing the Catalog Columns

The **Catalog** tab displays the following information by default:

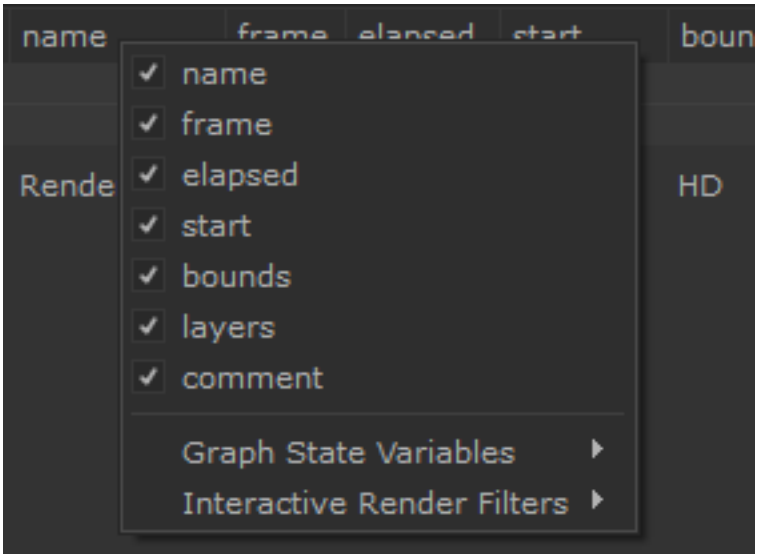
- **name** - The name of the node from where the render was started.
- **frame** - The frame time that the render was started.
- **elapsed** - The time taken for the render to complete (**h**:**mm**:**ss**).
- **start** - The time at which the render started.
- **bounds** - The full image resolution of the render.
- **layers** - The **Image Layers** (**AOVs**) that have been rendered.
- **GSVs** - The **Graph State Variables** used for the rendered image.
- **IRFs** - The **Interactive Render Filters** used for used for the rendered image.
- **comment** - Click to type a comment about a rendered image stored in the **Catalog** tab.



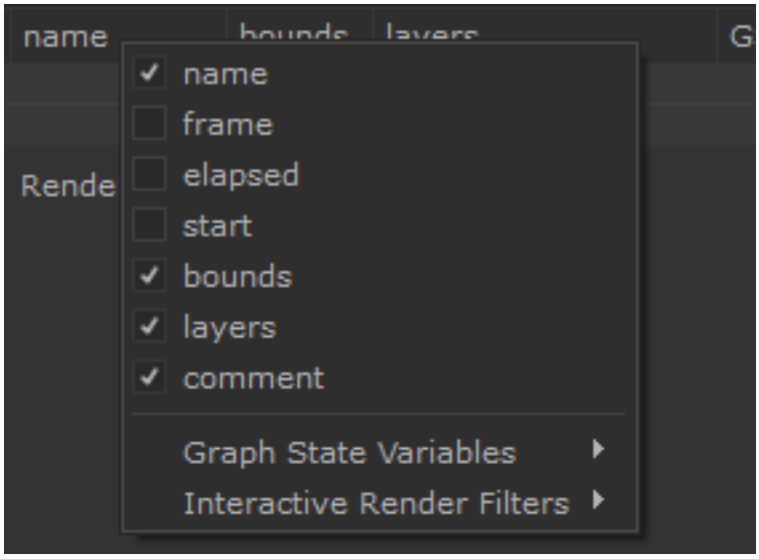
Note: For more information, see Manipulating Catalog Entries.

To customize what is displayed:

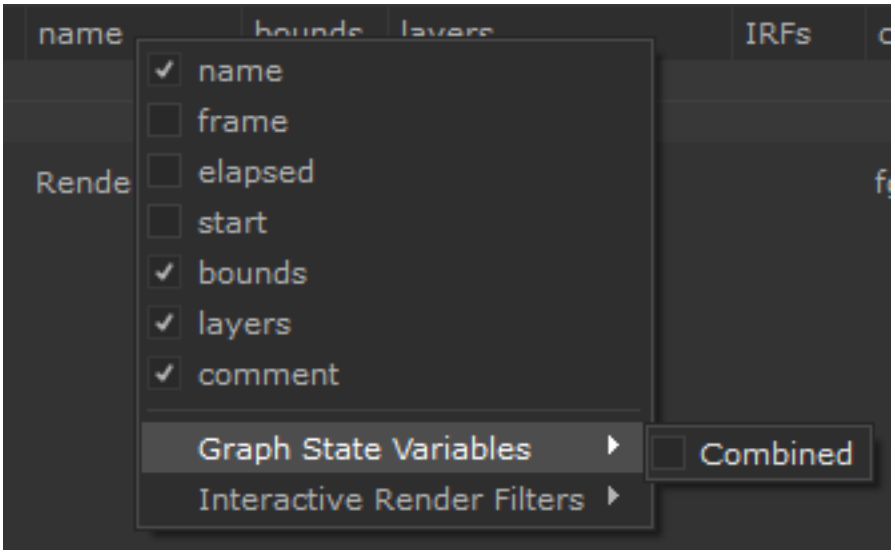
1. Right-click the header bar in the **Catalog** tab.



2. Deselect/select the required headers of your choice from the menu.



To disable the **GSVs** and **IRFs** columns, hover over the **Graph State Variables** or **Interactive Render Filters** option and deselect **Combined**.



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.

Manipulating Catalog Entries

Below are a list of features you can use to manipulate the Catalog entries:

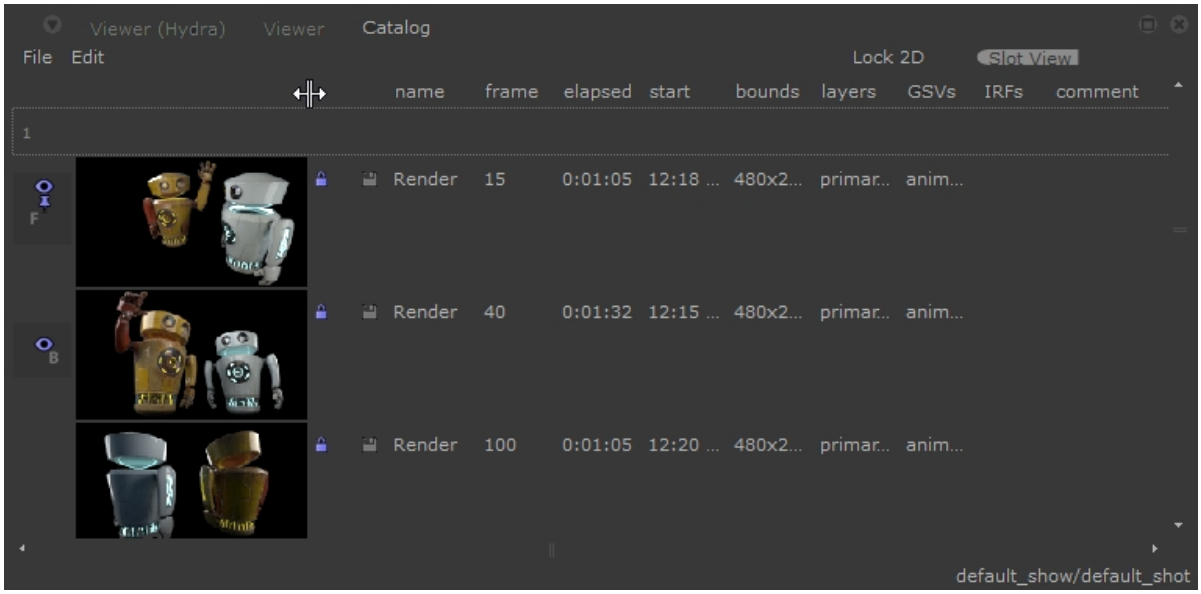
- To move Catalog entries from one slot to another, middle-click and drag.
- To toggle the lock status of a render, click / next to the image's thumbnail. Locked images are not overridden by a subsequent render to the same slot.
- To toggle whether an image is saved within this Catalog, click / next 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.
- To change the Region of Interest (ROI) to match the ROI of the render, right-click the area to the right of the render thumbnail, and select **Adopt Render ROI**.
- To change the current frame to match the frame of the render, right-click the area to the right of the render thumbnail, and select **Adopt Frame Time**.
- To select the node the Catalog render was generated from, right-click the area to the right of the render thumbnail, and select **Find in Node Graph**.
- To regenerate the thumbnail within the **Catalog** tab, right-click the area to the right of the render thumbnail, and select **Regenerate Thumbnail**.
- To create a copy of a Catalog item, right-click the area to the right of the render thumbnail, and select **Duplicate Catalog Item**.
- To make 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.
- To toggle 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 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,
- Right-click a thumbnail to make it the **Back** image, or
- **Ctrl**-click a thumbnail to pin the image to the **Front** buffer. Pinning a thumbnail prevents subsequent renders updating the **Front** buffer.

Click the thumbnail again to unpin the image.



Importing and Exporting Catalog Entries

Importing an Image or File Sequence to the Catalog

1. Select **File** > **Import Image / Sequence** (from the **Catalog** tab).

   The **Import Image / Sequence** dialog 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**.

Note: During import only a single AOV, the primary pass, is read into the Catalog and becomes available for preview within the **Monitor** tab.

Exporting an Image or File Sequence from the Catalog

1. Select **File** > **Export Catalog** and:
   - **All Images** - all entries in the **Catalog** tab.
   - **Locked Images** - only those entries in the **Catalog** tab that are locked.

- **Selected Images** - only selected entries in the **Catalog** tab.

The **Export Catalog** dialog displays.

2. Choose one of the export methods below:
   - Specify the full path to the directory where your files are exported. File names are automatically generated and begin with **catalog_export**.
   - Specify the full path for a sequence. The sequence must follow the pattern **sequenceName.#.fileFormat**, where **#** is the amount of padding for image numbers.

   The supported export formats are **.exr**, **.tif**, **.jpg**, and **.png**.
3. Click **Accept**.



Note: When manually entering in a directory path, you must include the file path and name of the directory in order to click **Accept**.

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** on the keyboard).
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.

Viewing the Render Log for a Catalog Entry

The **Render Log** 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 **Render Log**.

To view a **Catalog** entry's **Render Log** output, click its thumbnail. The entry becomes the **Front** render in the **Monitor** tab and its **Render Log** entry is displayed within the **Render Log** 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.

To view 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> .

To change 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.

To change the scale of the plotted values in the Y axis, enter a new value within the **vScale** field.

To toggle a channel's 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.



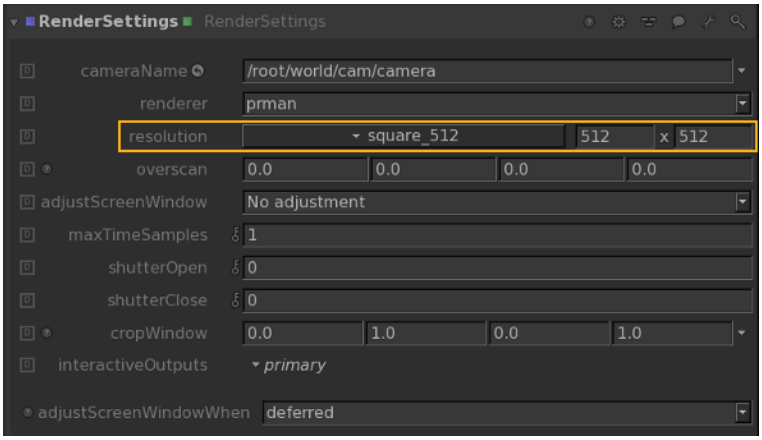
Note: The plotted value does not necessarily correspond to an actual value. For instance, the range for the **lnf** colorspace within the **Histogram** tab is 0 to 1023 whereas the actual values are 32-bit floating point. Katana maps the colorspace values to a range for display purposes.

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.

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.

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.
- Renderer-specific node types, such as **DlSettings** or **PrmanObjectStatements**, can be used to fine-tune renderer-specific configuration. These nodes typically set attributes on relevant scene graph locations, such as the targeted object, or **/root** for global settings.

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.

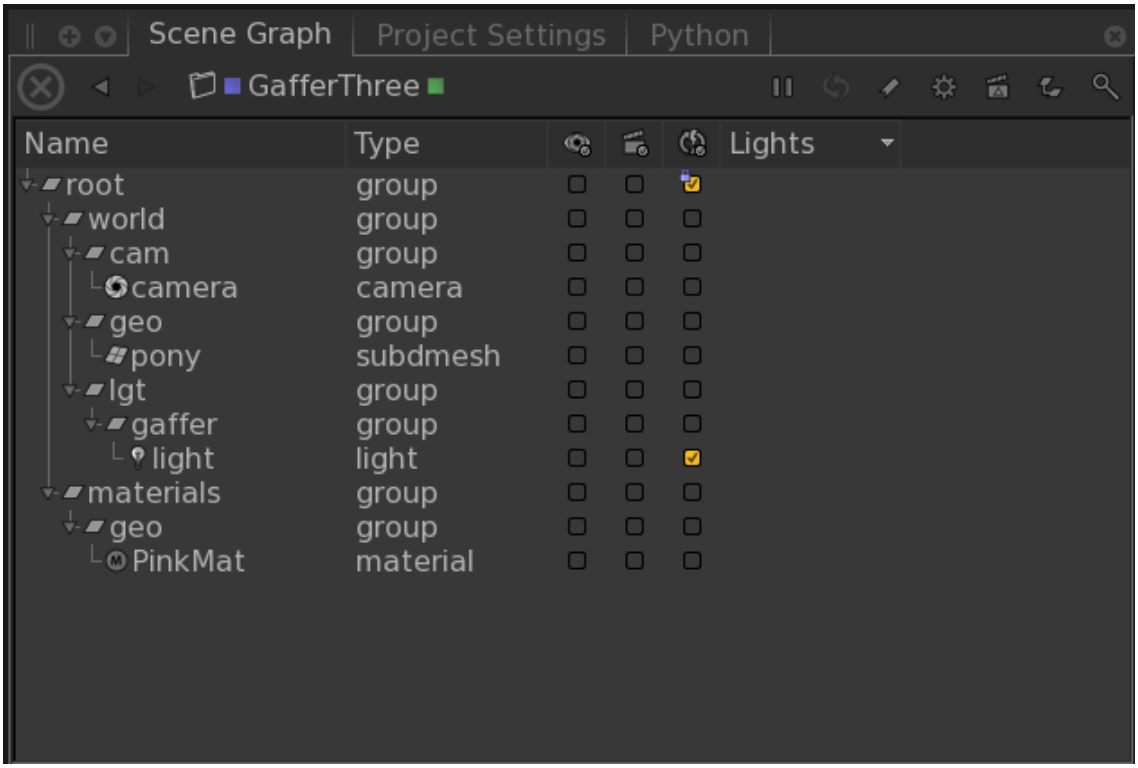
   The image in the **Monitor** tab is updated in response to your actions in the **Viewer** tab.

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.

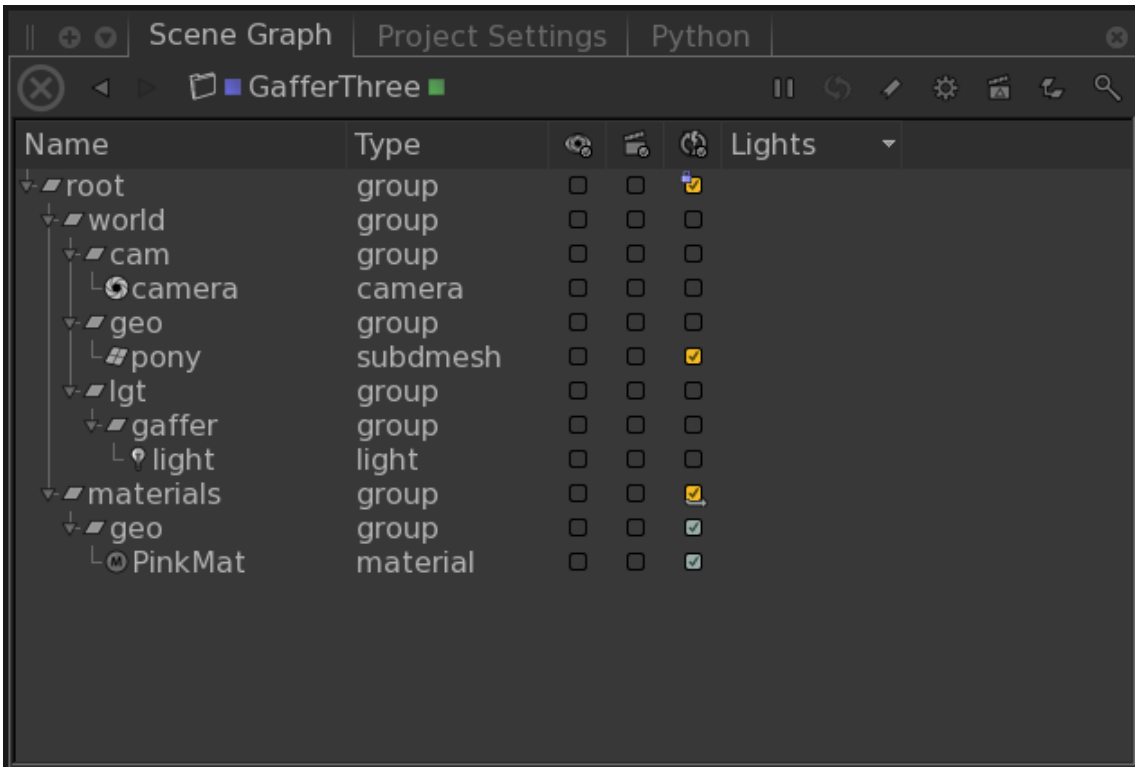


Note: If a light is to be used as the Live Render camera, it is enabled in the **Scene Graph Live** column.

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 node parameter values are finalized with all pending changes prior to performing a render.

3D Update Mode on	Indicators
Manual mode	
Pen-up mode	
Continuous mode	

Note: Whether or not changes to the node graph topology are supported by a renderer plug-in depends on the capabilities of that plug-in or the renderer that it represents.

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.

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

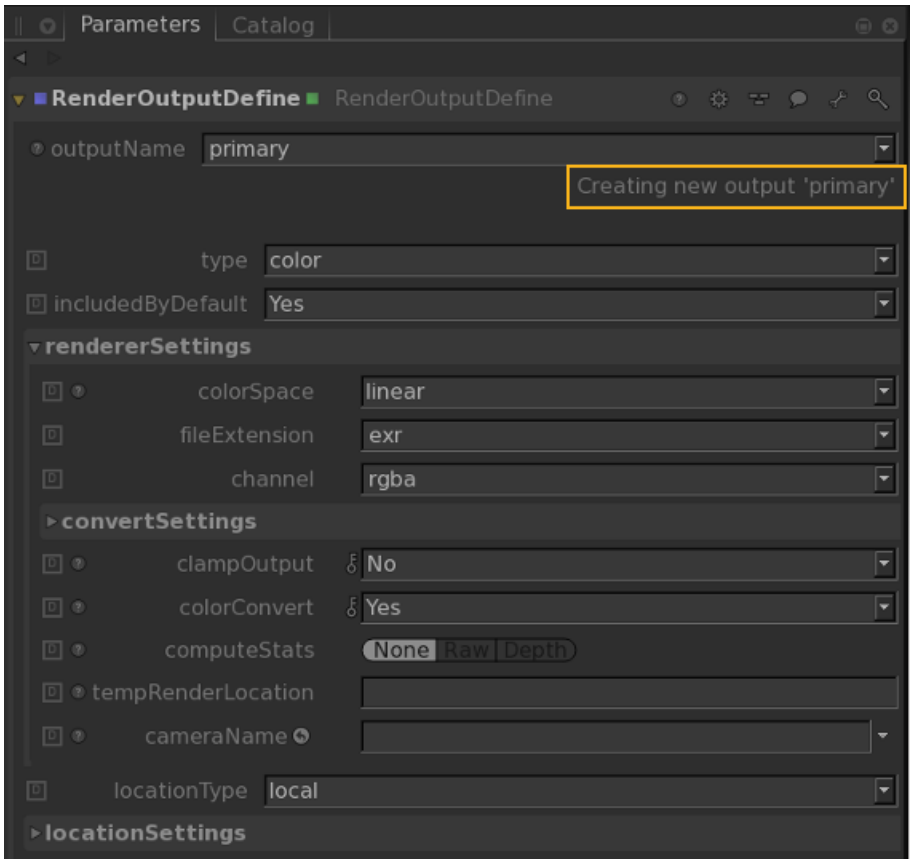
The RenderOutputDefine node can be used to create a new render output or override the settings for an existing one.

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.

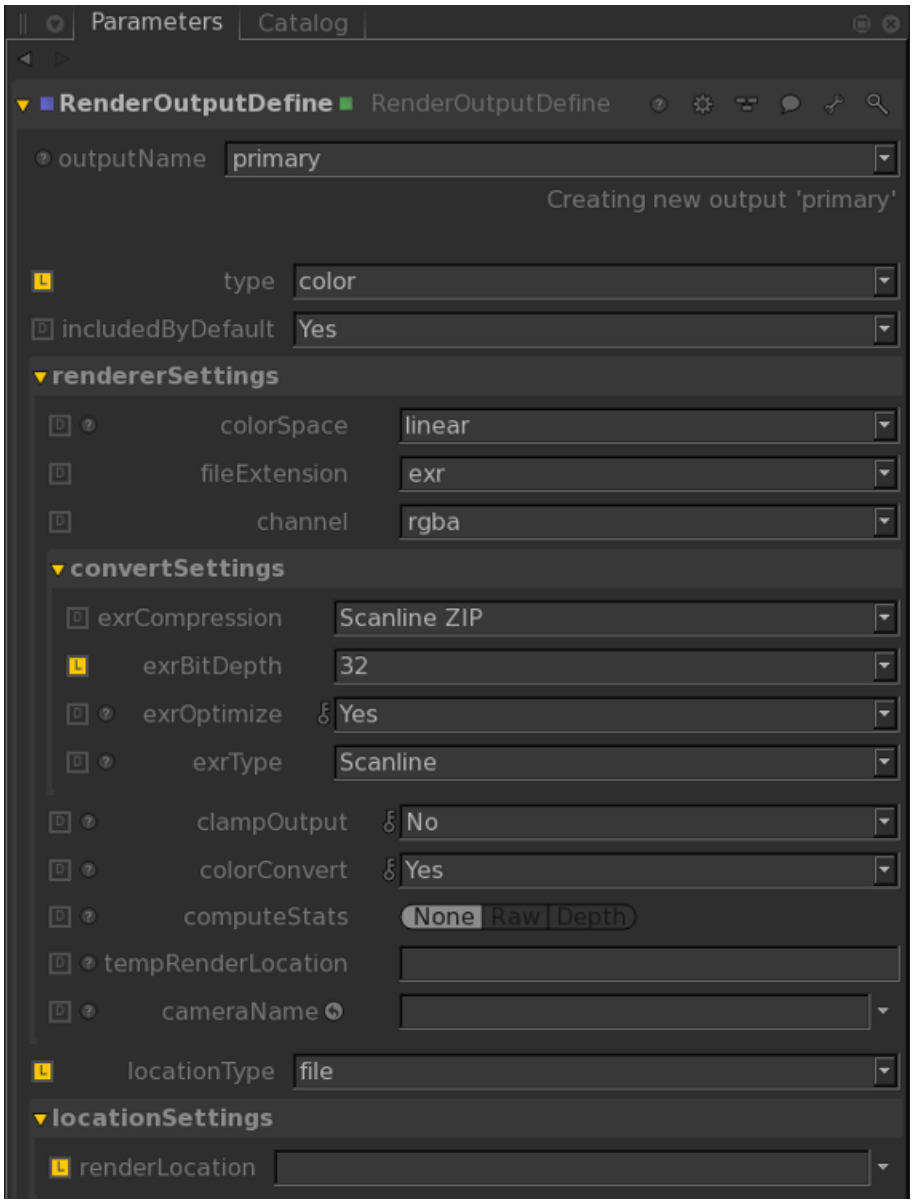
The output colorspace is ignored if the **colorConvert** dropdown is set to **No**. For more on colorspaces within Katana, see Managing Color.

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, certain colorspaces require 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 table lists the supported outputs for 3Delight, PRMan, and Arnold.

3Delight	PRMan	Arnold	Description	
color	**color**	**color**	Used for most renders.	
	deep	**deep**	Used for deep **.exr** creation in RIS workflows, and for deep shadow map creation in legacy REYES workflows.	
	raw	**raw**	Used when no color management is needed.	
	script	**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	**prescript**	Used to inject a command-line script into the render process that runs before the render is started.	
	merge	**merge**	Used to merge a number of render outputs (usually AOVs) into a single OpenEXR file.	
	none	**none**	Clears the pass, removing it from the output list.	

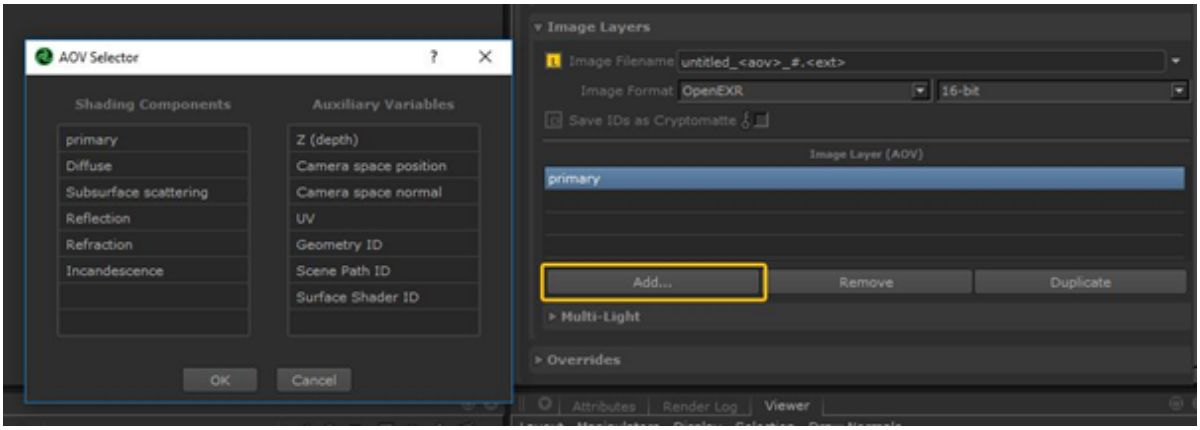
Defining an AOV Output

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 little extra processing cost. The ability to define AOVs is fully supported in Katana and is easy to set up.

Defining and Rendering AOVs with 3Delight

The 3Delight for Katana plugin includes the DlSettings node type, which is a SuperTool that allows you to define AOVs as well as adjustments to the render settings.

1. While editing the parameters of a DlSettings node in the **Parameters** tab, navigate to the **Image Layers** section and click the **Add...** button to open the **AOV Selector** window.



2. Under **Shading Components**, select **Reflection**, and under **Auxiliary Variables** select **Camera space position** and click **OK**.
3. In the **Node Graph** tab, right click the DlSettings node and click **Preview Render**.
4. Once the render has completed, in a **Monitor** tab, click the text **default** for a list of available preview passes.



Note: You will notice that while the **Reflection** pass displays properly, there is no information from the **Camera space position** pass. This is because shading component passes get their information from material nodes upstream of the DlSettings node, but Auxiliary Variables need to have their source from the renderer itself.

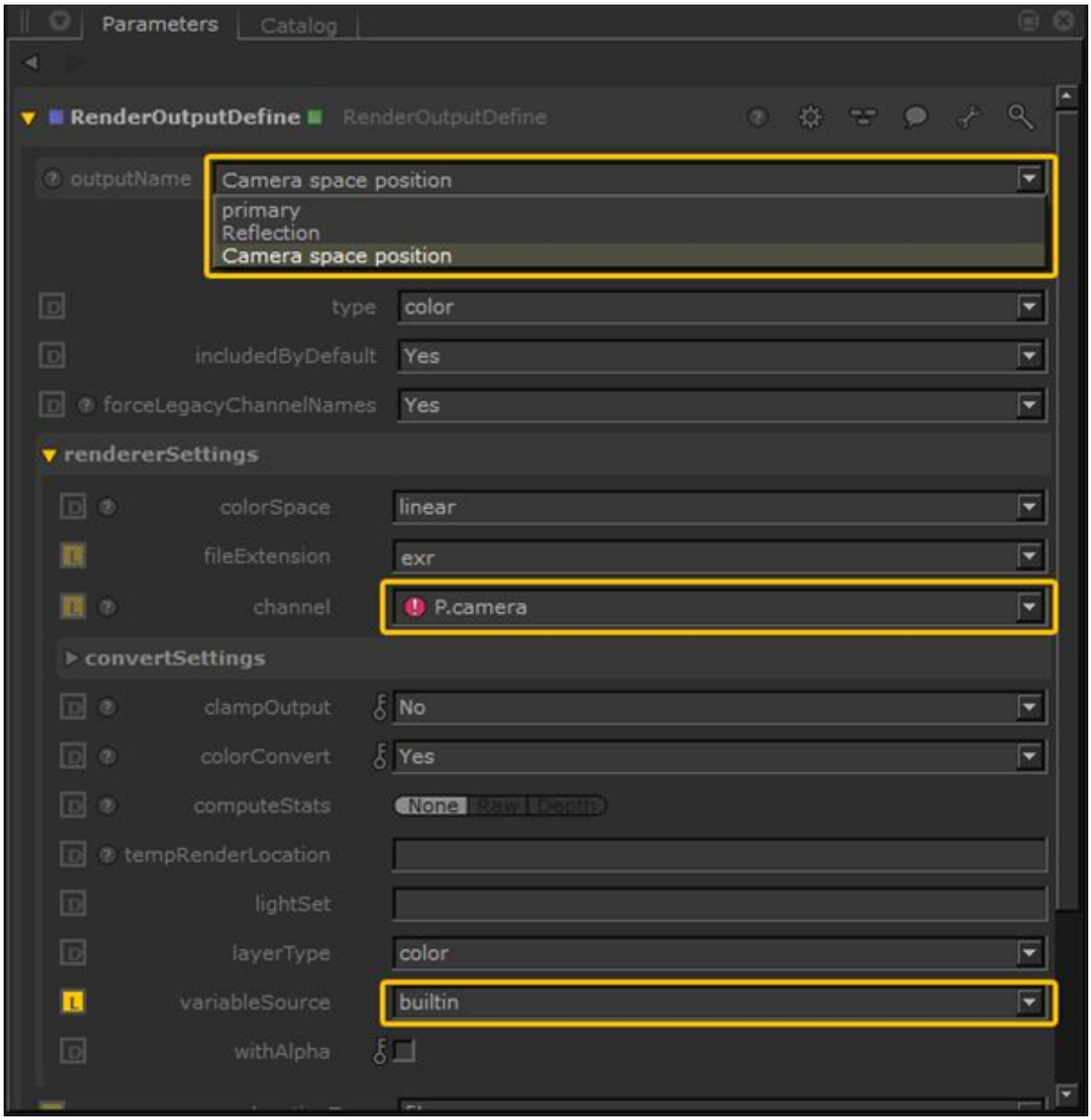


5. Click on a pass name to change what you see in the **Monitor** tab.

> **Tip:** You can cycle through the AOVs in the monitor using **Shift+Page Down** to move through the list from first to last, and **Shift+Page Up** to move from last to first. You can also use **Shift**+**Home** to toggle between the default AOV and the last selected AOV.

6. In the **Node Graph** tab, add a RenderOutputDefine node and connect it to the output of the DlSettings node. Set the node's edit flag to bring up its parameters in the **Parameters** tab.
7. Set the **outputName** parameter to **Camera space position**. The **channel** value changes automatically to match the selection.

8. Change **variableSource** from **shader** to **builtin**.



9. Preview render from this node. You will now be able see the position pass render in the **Monitor** tab.

Defining and Rendering AOVs with Other Renderers

1. Add a RenderOutputDefine node.
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. See the Reference Guide for details on the various parameters.



Note: Notes: Please refer to your chosen renderer's documentation for names and definitions of AOV channels.

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 An Output Pass.



Warning: Progressive interactive renders, when configured to send image updates (buckets) with high frequency, may flood the message queue of the renderer plug-in's display driver. To prevent this from consuming unreasonable amounts of memory, the queue is limited in size and, when full, results in delays in updates being sent to Katana. A warning is then printed to the Render Log. The size of the queue (as a number of messages) can be specified using the environment variable **KATANA_PIPE_MAX_QUEUE_SIZE**. The default size is 16384.

Instancing

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 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.

The greatest benefits from instancing are seen when rendering many instances of complex source geometry. As mentioned previously, 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.

Rendering Instances

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**.



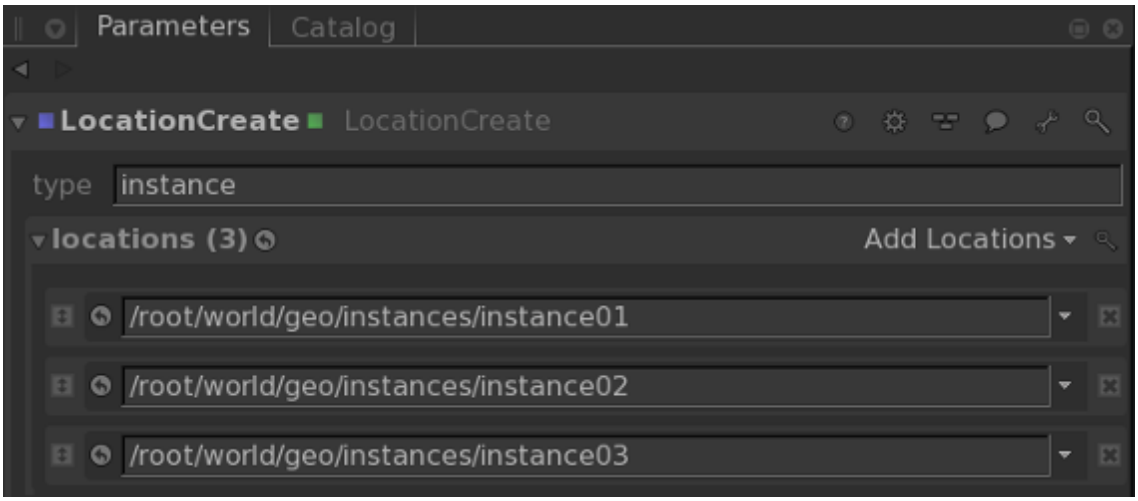
Note: The scene graph location of the instance source is wrapped in a Katana procedural (• **katana**). This is a render agnostic workflow that will also work with Prman and Arnold.

The workflow associated with your preferred renderer may be more efficient. Check their documentation for details.

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 OpScript node, and set the CEL field to collect all instance locations by choosing **Add Statements** > **Custom** and entering - in this case - **/root/world/geo/instances//***

   Enter the following in the OpScript node's **script** parameter to set each instance location's **geometry.instanceSource** attribute (in this case **/root/world/geo/instanceGroup**):

```
local source = "/root/world/geo/instanceGroup"

local attributeName = "geometry.instanceSource"

local attributeValue = StringAttribute(source)

Interface.SetAttr(attributeName, attributeValue)
```

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 at render time that 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 AttributeSet 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 [mix x, max x, min y, max y, min z, max z]

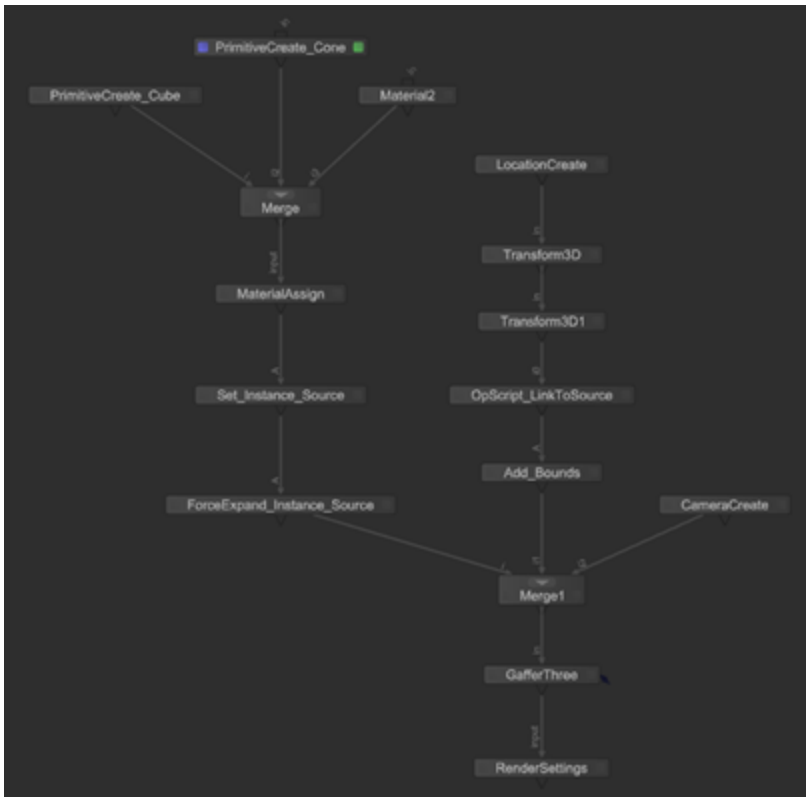
3. Add an AttributeSet node downstream of the two PrimitiveCreate nodes, and point it to the instance source location.

   Set the **attributeName** parameter to **forceExpand**, the **attributeType** to **integer**, and the **numberValue** to **1.0**.

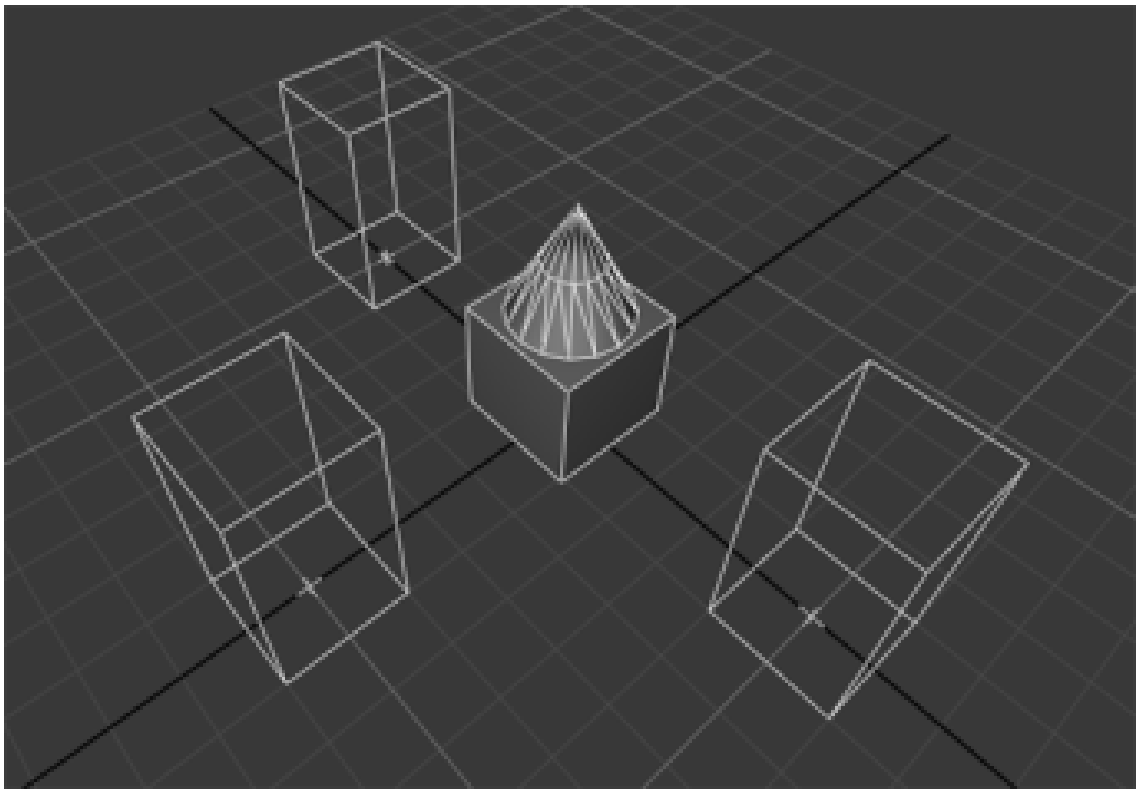
Render the Scene to See the Instances

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.

> **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.

Writing EXR Metadata from Katana

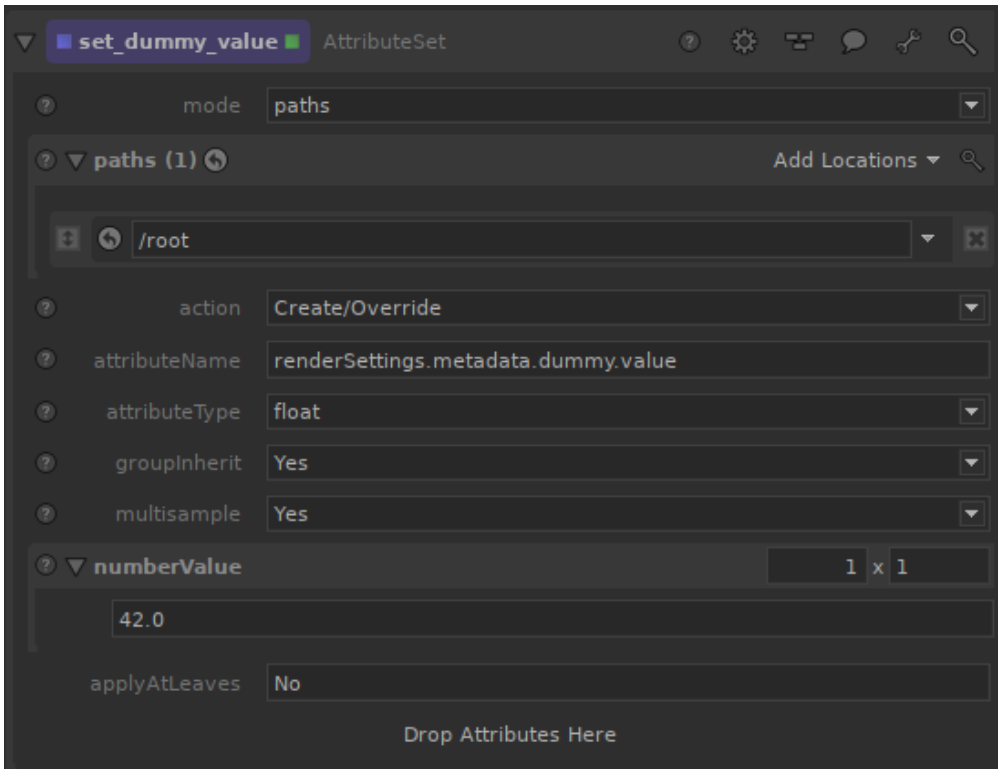
It is possible to create custom EXR metadata fields in Katana disk renders in both Arnold and RenderMan. This can be created using **AttributeSet** nodes.

In the **AttributeSet** node, set the **attributeName** at **/root** to **renderSettings.metadata.<attrib>.[key|value|prefix]** where <attrib> is an arbitrary attribute identifier.

You will need to include key or value to output metadata to the EXR.

Note: If there is no prefix, the property will be **exr/katana:<key> = <value>**. If a prefix is specified, the property will be **exr/<prefix>:katana:<key> = <value>**.

For example, setting a value (with 'dummy' as the attribute identifier):



OpenEXR Header Metadata

Warning: This is only currently supported by RenderMan.

An alternative method for adding arbitrary metadata to OpenEXR headers for RenderMan is to use the **OpScript** node. To do this, you can create attributes under **exrheaders** and use the **OpScript** node targeting the **/root** location to set the following:

```
local EXR_String =
"renderSettings.outputs.primary.rendererSettings.exrheaders.test_string"
local EXR_String_Value = StringAttribute("A String")
Interface.SetAttr(EXR_String, EXR_String_Value)

local EXR_Integer =
"renderSettings.outputs.primary.rendererSettings.exrheaders.test_int"
local EXR_Integer_Value = IntAttribute(1)
```

```
Interface.SetAttr(EXR_Integer, EXR_Integer_Value)

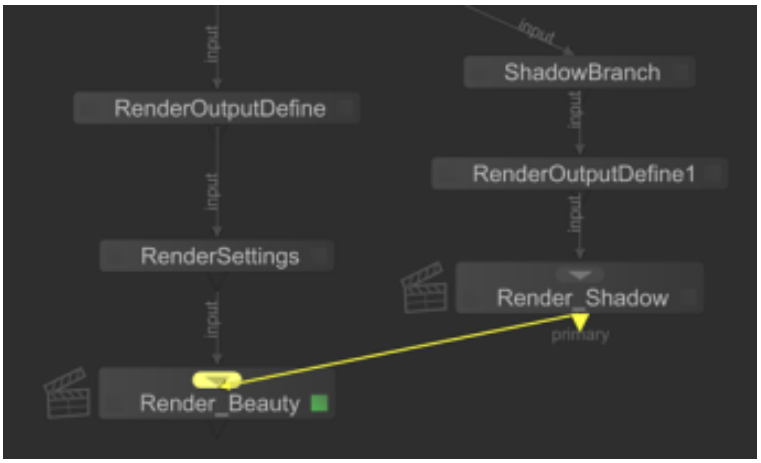
local EXR_IntegerArray =
"renderSettings.outputs.primary.rendererSettings.exrheaders.test_intArray"
local EXR_IntegerArray_Value = IntAttribute({1,2,3,4})
Interface.SetAttr(EXR_IntegerArray, EXR_IntegerArray_Value)

local EXR_Float =
"renderSettings.outputs.primary.rendererSettings.exrheaders.test_float"
local EXR_Float_Value = FloatAttribute(1.5)
Interface.SetAttr(EXR_Float, EXR_Float_Value)

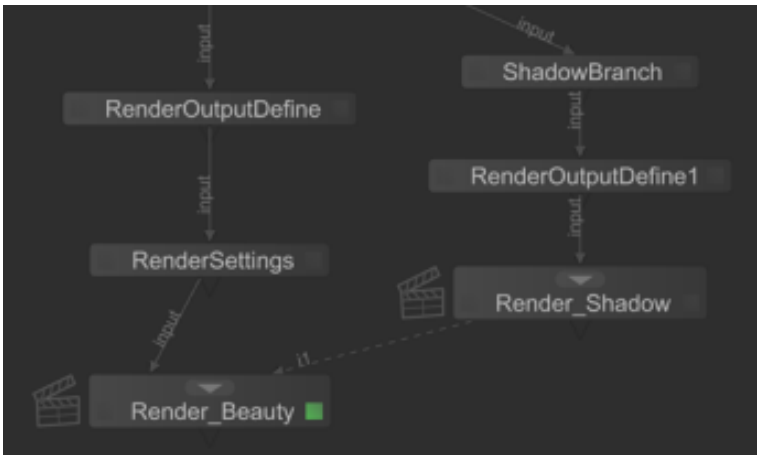
local EXR_FloatArray =
"renderSettings.outputs.primary.rendererSettings.exrheaders.test_floatArray"
local EXR_FloatArray_Value = FloatAttribute({2.6,3.8})
Interface.SetAttr(EXR_FloatArray, EXR_FloatArray_Value)
```

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.



Batch Mode

Batch mode allows you to render sequences of frames from a Katana scene all at once. It is started through a command line, where you specify the file path, frame range and any other necessary options.

Note: You will only be able to access terminal modes, including Batch Mode, if you have a Katana render license (katana_r). If you're a student, you can access one for free.

Batch mode is useful if you have a large number of frames to render as it will render out each individual file in the background. You can continue working on a Katana scene file whilst it is being batch rendered as the command uses the last saved version.

Before starting a batch render, ensure the render settings and the render flag are all set up correctly in Katana. To set the render flag, select the node you wish you render from and press **V** on the keyboard. The render flag can be determined through the command line, however setting it up beforehand simplifies the string needed to run Batch mode and minimizes any room for error.

Note: When you specify the Image Filename for the output render, ensure you use one or more hashes as they will be replaced by the frame number in your rendered file name. For example: fileName_<aov>_###.<ext>

Start a Batch Render

Windows

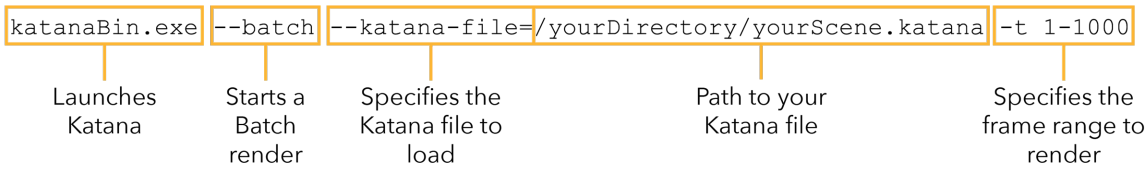
1. Open the Command Prompt.
2. Navigate to the directory where you have Katana installed using the cd command, for example:

```
cd C:\Program Files\Katana3.2v1\bin
```

3. Enter the following command to start a batch render:

```
katanaBin.exe --batch --katana-file=C:\yourDirectory\yourScene.katana
-t 1-1000
```

Where:



4. Press **Enter** to start the render.

You can add more arguments to the command. For example, use **--render-node** to specify the node you would like to render from if you haven't set your render flag in the Katana scene or if you would like to change it:

```
katanaBin.exe --batch --katana-file=C:\yourDirectory\yourScene.katana --
render-node=renderHere -t 1-1000
```

Linux

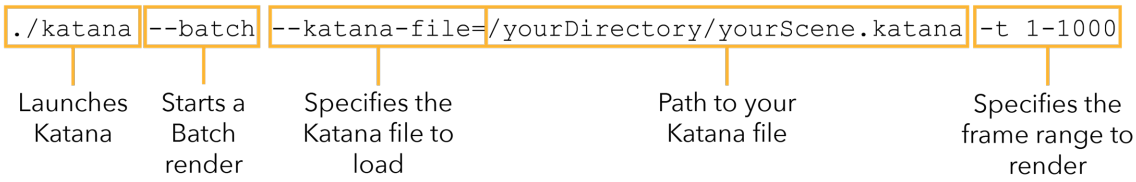
1. Open a Terminal.
2. Navigate to the directory where you have Katana installed using the cd command, for example:

```
cd /opt/foundry/katana
```

3. Enter the following command to start a batch render:

```
./katana --batch --katana-file=/yourDirectory/yourScene.katana -t 1-
1000
```

Where:



4. Press **Enter** to start the render.

You can add more arguments to the command. For example, use **--render-node** to specify the node you would like to render from if you haven't set your render flag in the Katana scene or if you would like to change it:

```
./katana --batch --katana-file=/yourDirectory/yourScene.katana --render-node=renderHere -t 1-1000
```

Here is a full list of command line options for Batch Mode:

Option	Usage

--katana-file	Specifies the Katana recipe to load.
	
	Syntax:
	`--katana-file=<filename>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000`
	`--render-node=beauty`

--asset	Specifies the asset ID to resolve.
	
	Syntax:
	`--asset=<asset ID>`
	
	Example:
	`./katana --asset=mock:///show/shot/name/version`

-t or --t	Specifies the frame range to render.
	
	Syntax:
	`-t <frame range>`
	
	OR
	`--t=<frame range>`

Option	Usage
	Where <frame range> can take the form of a range (such as 1-5) or a comma separated list (such as 1,2,3,4,5). These can be combined, for instance: 1-3,5, which would render frames 1, 2, 3, and 5. Example: `./katana --batch --katana-file=/tmp/test.katana` **`--t=1-5,8`** `--render-node=beauty`

--var	Sets the value of an existing Graph State Variable. This command-line option can be specified multiple times to override the values of multiple Graph State Variables.
	
	Syntax:
	`--var <GSV name>=<GSV value>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1 --var Shot=Sh1 --var timeOfDay=night --var variant=B --render-node=beauty`

--threads2d	Specifies the number of additional processors within the application. An additional processor is also used for Katana's main thread.
	
	This means that Katana uses 3 processors when **--threads2d=2**.
	
	Syntax:
	`--threads2d=<num threads>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000` **`--threads2d=2`** `--render-node=beauty`

--threads3d	Specifies the number of simultaneous threads the renderer uses.
	
	Syntax:
	`--threads3d=<num threads>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000` **`--threads3d=8`** `--render-node=beauty`

Option	Usage
--render-node	Specifies the Render node from which to render the recipe. Syntax: `--render-node=<node name>` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000` **`--render-node=beauty`**
--render-internal-dependencies	Allows any render nodes that don't produce **asset** outputs to be rendered within a single katana **--batch** process. Asset outputs are determined by asking the current asset plug-in if the output location is an assetId, using **isAssetId()**. The default **file** asset plug-in that ships with Katana classes everything as an asset. So at present it is not possible to render any dependencies within one katana **--batch** command without customizing the asset plug-in.

--crop-rect	Specifies which part of an image to crop. The same cropping area is used for all renders.
	
	Syntax:
	`--crop-rect="(<left>,<bottom>,<width>,<height>)"`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000`
	`--render-node=beauty` **`--crop-rect="(0,0,256,256)"`**

--setDisplayWindowToCropRect	Sets the display image to the same size as the crop rectangle set by **--crop-rect**.
--tile-render	Used to render one tile of an image divided horizontally and vertically into tiles. For instance, using **--tile-render=1,1,3,3** splits the image into 9 smaller images (or tiles) in a 3x3 square and then renders the middle tile as the index for tile renders starts at the bottom-left corner with 0,0. In the case of 3x3 tiles, the indices are: 0,2 1,2 2,2

Option	Usage
	0,1 1,1 2,1 0,0 1,0 2,0 The results are saved in the same location as specified by the RenderOutputDefine node but with a tile suffix. For instance: tile_1_1.beauty.001.exr Syntax: `--tile-render=<left_tile_index>, <bottom_tile_index>, <total_tiles_width>, <total_tiles_height>` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=0,0,2,2`** `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=0,1,2,2`** `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=1,0,2,2`** `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--tile-render=1,1,2,2`**

--tile-stitch	Used to assemble tiles rendered with the **--tile-render** flag into a complete image.
	
	When stitching, you must still pass the **--tile-render** argument, with the number of x and y tiles, so that the stitch knows how many tiles to expect, and their configuration.
	
	Syntax:
	`--tile-render=<left_tile_index>, <bottom_tile_index>, <total_tiles_width>, <total_tiles_height> --tile-stitch`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana -`

Option	Usage
	`-t=1-1000 --render-node=beauty --tile-render=0,0,2,2` **`--tile-stitch`**
--tile-cleanup	Used to clean up transient tile images. Can be used in conjunction with **--tile-stitch** to assemble a complete image, and remove transient tiles in a single operation. When using **--tile-cleanup** you must still pass the **--tile-render** argument with the number of x and y tiles, so that cleanup knows how many tiles to remove. Syntax: `--tile-render=0,0,<total_tiles_width>,<total_tiles_height> --tile-cleanup` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty --tile-render=0,0,2,2 --tile-stitch` **`--tile-cleanup`**

--prerender-publish	In Batch mode, it executes the **Pre-Render Publish Asset** action on the outputs but doesn't render images.
	
	The value specifies the filename for dumping render pass information.
	
	Note: This can be used together with `--versionup`.
	
	Syntax:
	`--prerender-publish=<pass info>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--prerender-publish`**`=/tmp/pass_info.xml`

--make-lookfilebake-scripts	Used to write out a number of Python files that can be executed in Batch mode to write look files. Syntax:

Option	Usage

	`--make-lookfilebake-scripts=<script directory>`
	
	Example:
	`./katana --batch --katana-file=/tmp/bake.katana --t=1`
	`--make-lookfilebake-scripts=/tmp/bake_scripts`
	`./katana --script /tmp/bake_scripts/preprocess.py`
	`./katana --script /tmp/bake_scripts/lf_bake_default.py`
	`./katana --script /tmp/bake_scripts/postprocess.py`

--postrender-publish	In Batch mode, it executes the **Post-Render Publish Asset** action on the outputs but doesn't render images.
	
	The value specifies the filename for dumping render pass information.
	
	  **Note:** This can be used together with `--versionup`.
	
	Syntax:
	`--postrender-publish=<pass info>`
	
	Example:
	`./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--postrender-publish`**`=/tmp/pass_info.xml`

--versionup	Used to specify that you want to version up assets when publishing to the asset management system. Syntax: `--versionup` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--versionup`**
--reuse-render-process	Iterates over the sequence of frames to render, and exports Op

Option	Usage
	Tree files for all frames, then starts the renderer (/renderboot process) only once on a sequence of exported Op Tree files. Syntax: `--reuse-render-process` Example: `./katana --batch --katana-file=/tmp/test.katana --t=1-1000 --render-node=beauty` **`--reuse-render-process`**



Note: Setting **threads3d** or **threads2d** through Batch mode takes precedence over the **interactiveRenderThreads3D**, and **interactiveRenderThreads2D** settings in Katana's **Edit** > **Preferences** > **application** menu.



Article: How to render an image in multiple tiles in Batch Mode.

Advanced Workflow & Extensions

Nuke Bridge

Katana's Nuke Bridge lets you stream a render to Nuke for compositing, and then streams the comped pixels back to Katana for review or refinement.

Asset Management

An asset is an item of data that contributes to a Katana project.

LiveGroups

How to import another Katana project in to the current project and reload it every time the current project is updated.

Graph State Variables

Control which nodes in the node graph contribute to scene graph processing, based on the values of user-set variables.

Scripting and Programming

Get started scripting and programming in Katana using Python, Lua or C++.

Groups, Macros and Super Tools

An introduction to Groups, Macros and Super Tools.

Customizing GafferThree

Learn how to create custom package classes for GafferThree node types.

See a Nuke Comp of Your Project in Katana Using the Nuke Bridge

Why Get a Comp from Nuke?

It's important for artists to be able to assess that Katana scenes are accurate for the next stages in the pipeline. However, as the work progresses down the pipeline and iterations are applied, elements of the scene such as lighting or materials may need to be changed to make them fit more accurately with the results of the composited image.

This process is usually done by producing offline/disk renders from Katana and passing them on to the next stage of the pipeline. Only once these renders were imported and set up in Nuke, would Artists be able to review the results. In some cases, this is a lengthy and manual process, adding time to the overall production process.

Katana's Nuke Bridge is a game-changing feature that lets you stream a render to Nuke for compositing, and then streams the comped pixels back to Katana for review or refinement. You see the results of the comp within Katana's render window, eliminating the need for viewing offline and waiting for renders.



In the Nuke Bridge a Katana render is streamed to Nuke, composited using a Nuke script, and the result streamed back to Katana as a new render.

What Do I Need to Use Nuke Bridge?

To get Nuke Bridge up and running you'll need the following:

- Katana 4.5 or later.
- Nuke 13.0 or later.
- A nuke_i (interactive) and nuke_r (render) license to run all three comping modes, though a nuke_i license can support all three with additional configuration. See below for details on license compatibility.
- A custom Nuke launcher script with a variable set to load the Katana Nuke plug-ins.
- A Nuke script containing KatanaReader and KatanaWriter nodes.
- A Katana launcher script with a variable set to your version of Nuke.

Full instructions for creating the scripts are given in the following sections.

You can download a Nuke test script and a simple Katana project to try the Nuke Bridge from here.

Nuke and Katana are available for download on the following pages:

- Download Nuke from here.
- Download Katana from here.

Which Nuke Licenses do I Need to Use Nuke Bridge?

The table below shows you which comping modes are available depending on your Nuke license, nuke_i (interactive GUI) or nuke_r (render only).

	nuke_i and nuke_r	nuke_i	nuke_r
Preview Comp	Yes	Yes (with extra configuration*)	Yes
Live Comp	Yes	Yes (with extra configuration*)	Yes
Interactive Comp	Yes	Yes (with extra configuration*)	No

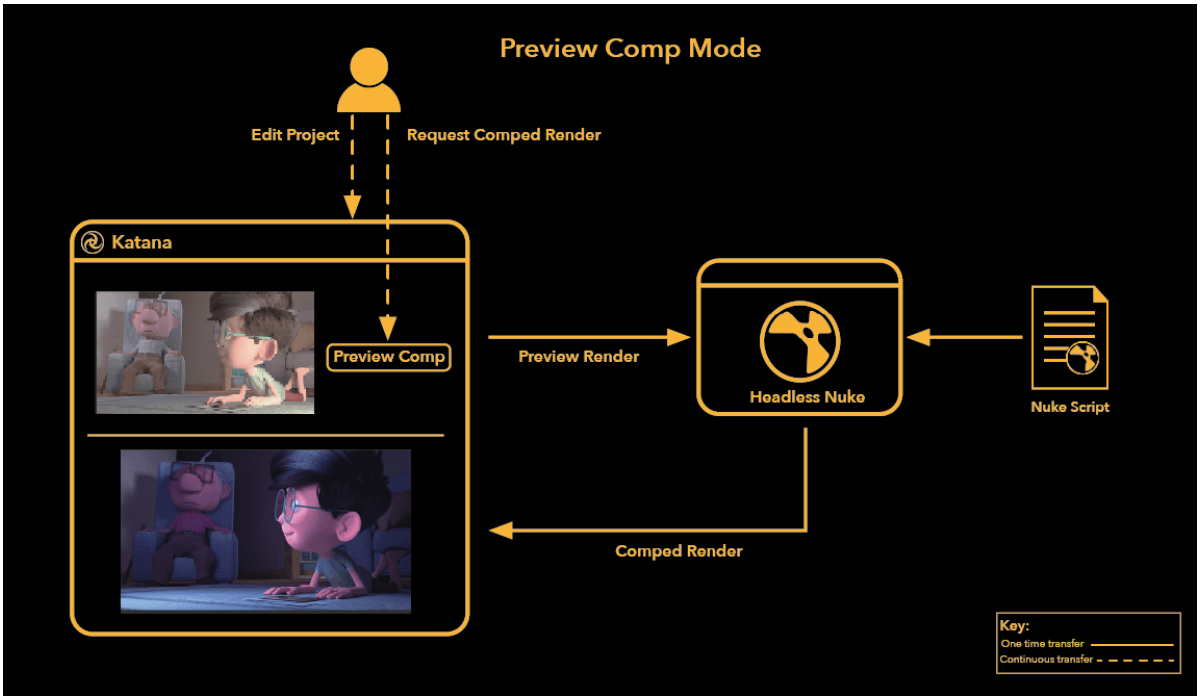
Note: *If you only own a nuke_i license, you need to create a custom launcher script for Nuke and customize the Katana launcher script to reference this additional Nuke launcher script. Instructions on how to write a launcher script for a nuke_i license are outlined in Additional Steps for Setting up the Nuke Bridge With Only a nuke_i License, however you should still follow all preceding set-up instructions given in the following sections.

- To check which licenses you have, user the Foundry Licensing Utility, available here.

Choose How to Work With Nuke Using Three Comp Modes

Nuke Bridge offers three modes of operation depending on how you want to work with your render and the comp.

Preview Mode

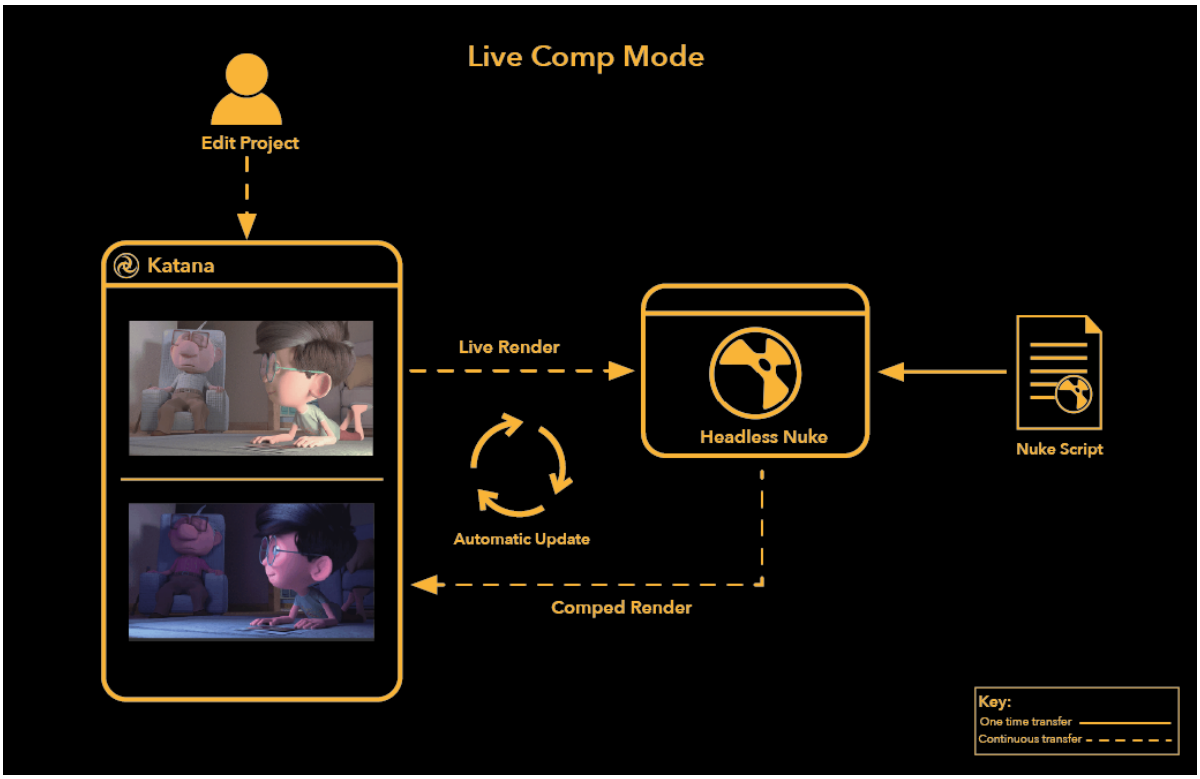


Preview mode allows you to send preview renders from Katana, to a headless version of Nuke and then view a composited version of the preview render within Katana, allowing for a quick snapshot of how your final image may look. Whenever a new preview render is created, preview mode will need to be manually re-triggered so that the updated render is comped.

This mode runs Nuke as a background process, meaning you cannot interact with Nuke once the Preview Comp process has been triggered.

The version of Nuke launched by Preview Comp can be installed on a local machine or on a network.

Live Mode



Live mode allows you to send a live render from Katana to a headless version of Nuke, so a comped version of your live render can be viewed in Katana, while still being able to edit your lights and materials. Any edits to your live render are automatically streamed into Nuke and the comped render will update in real time.

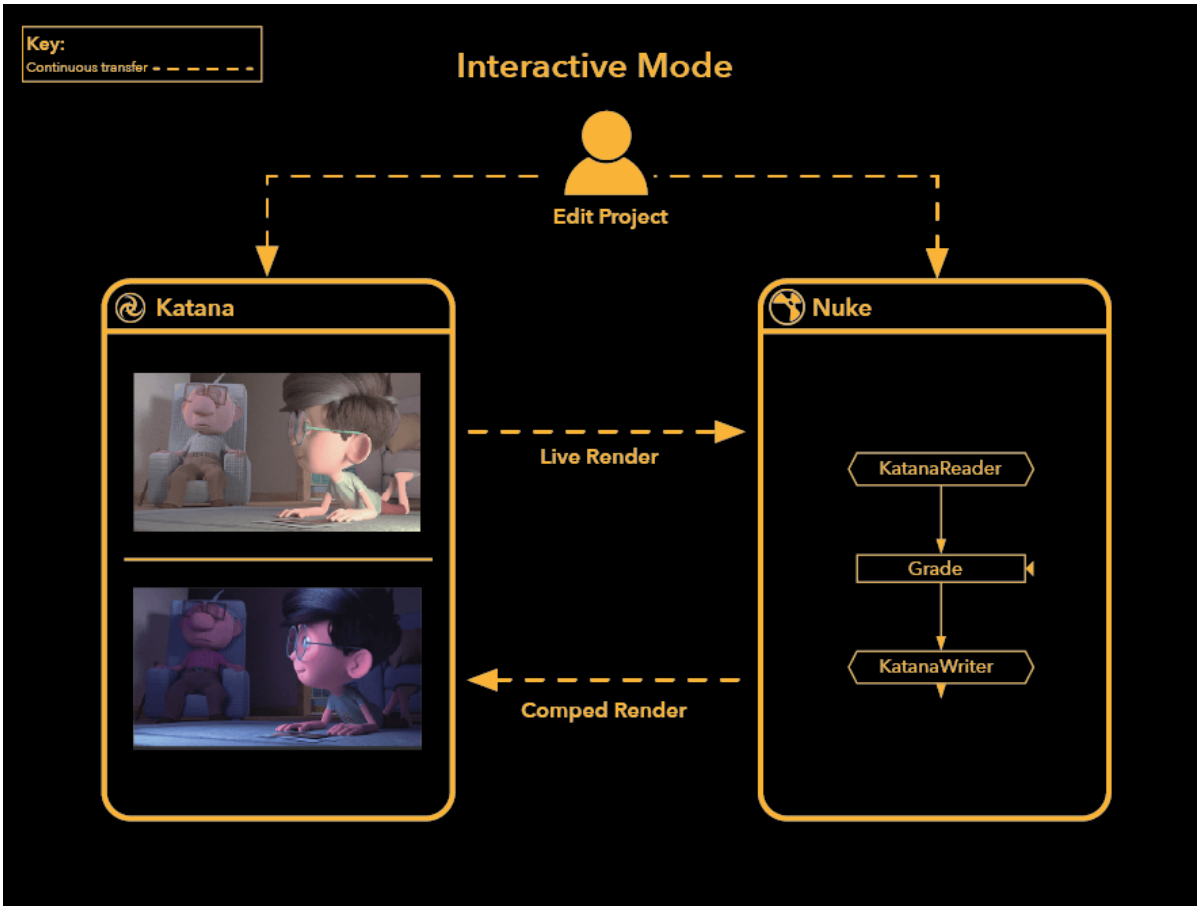
This allows you to be able to edit your scene in the context of a Nuke script and make sure that your lighting setup is compatible with a final grade or composite that your project may be using.

As with preview comp, Nuke is run as a background process and cannot be interacted with.

The version of Nuke launched by Live Mode can be installed on a local machine or on a network.

Note: For information on the differences between Live and Preview renders in Katana, see Rendering Types

Interactive Mode



Interactive mode can be used with Live or Preview renders, and when triggered the Nuke application opens. The most flexible of the three modes, Interactive allows you to continue to edit your Katana scene, whilst also being able to adjust your Nuke script within Nuke.

When working with live renders, edits made in Katana are automatically streamed to Nuke, while edits to your Nuke script are automatically streamed back into Katana. When working with preview renders, any edits made to your scene need to be re-rendered and interactive mode needs to be re-triggered for the new render to be seen in Nuke. However, any changes to your Nuke script are reflected automatically in Katana, regardless of the type of render that is being streamed to nuke.

The version of Nuke launched by Interactive Mode must be installed locally and is not able to be used with versions of Nuke installed across a network

Set Up Nuke Bridge for Katana

This section assumes you're setting up Nuke Bridge on a single machine to work in any of the three comp modes. To set up Nuke Bridge across a network or render farm, each machine in your network will need to point to the same file path destinations set in the scripts and environment variables.

There are three main stages to the setup:

1. Configure a custom launcher script for Nuke to launch standalone so that you can create a compatible Nuke Script with the Katana plug-ins.
2. In Nuke, create a script including at least one KatanaReader and KatanaWriter node, which allows Nuke Bridge to stream renders to and from Katana.
3. Configure a custom launcher script to launch Katana with the correct settings.

Configuring a Nuke Launcher Script for the Katana Nuke Plug-in

For Nuke Bridge to work, you must create, or have access to, a Nuke Script that includes at least one KatanaReader and KatanaWriter node. To create these nodes in a standalone Nuke session, a custom launcher script should be created with a variable to define the location of the Katana Nuke plug-in. You can then use this script to run Nuke and add the Katana nodes.

The script performs the following actions:

1. Set the location of the Katana Nuke Plug-in
2. Launch Nuke
3. Allow users to create KatanaReader and KatanaWriter nodes in a Nuke script

Configure a Nuke Launcher Script for Windows

You can copy and edit the template from the code below:

Nuke Launcher script for Nuke Bridge

```
@echo off
set "NUKE_PATH=%NUKE_PATH%;C:\<PathToKatana>\<KatanaVersion>\plugins\Resources\Nuke\<NukeVersion>"
"C:\<PathToNuke>\<NukeVersion>\<NukeApplication>.exe"
```

To use KatanaReader and KatanaWriter nodes in Nuke standalone sessions, Nuke must be launched using a script that specifies the NUKE_PATH.

Tip: A quick way to get file paths is to navigate to the directory using your file explorer and copy the path from the explorer bar.

1. Edit the Katana installation location to include the versions of Katana and Nuke being used:

   ```
   set "NUKE_PATH=%NUKE_
   PATH%;C:\<PathToKatana>\<KatanaVersion>\plugins\Resources\Nuke\<NukeVersio
   n>"
   ```

   For example:

   set "NUKE_PATH=%NUKE_PATH%;C:\Program Files\Katana5.0v1\plugins\Resources\Nuke\13.1"

2. Edit the launcher variable With the version of Nuke and Nuke application you're using:

   ```
   "C:\<PathToNuke>\<NukeVersion>\<NukeApplication>.exe"
   ```

   This should match the version added to previous variable. For example:

   C:\Program Files\Katana5.0v1\plugins\Resources\Nuke\13.1

3. Save the file with a .bat extension.

   For example, **KatanaNuke.bat** would be an appropriate filename for a launcher script.

Configure a Nuke Launcher Script for Linux

You can copy the example below to create a template:

Nuke Launcher script for Nuke Bridge

```
#!/bin/bash

export NUKE_PATH="/<katana installation directory>/plugins/Resources/Nuke/<nuke version>"

"/<path to nuke installation directory>/<version of nuke>/<nuke application>"
```

Set an environment variable on Linux in the terminal, or in your own custom launcher script for Katana:

`export NUKE_PATH="/<katana installation directory>/plugins/Resources/Nuke/<nuke version>"`

For example, if you're running Katana 5.0v1 and Nuke 13.1:

export NUKE_PATH="/opt/Foundry/Katana5.0v1/plugins/Resources/Nuke/13.1"

If you're using a script, also edit the path to the Nuke application.



Warning: If you set the environment variable using the terminal, Nuke must launch from the same terminal.

Configure a Katana Launcher Script for Nuke Bridge

To use Nuke Bridge you need to launch Katana with a launcher script that sets the file path of the custom Nuke launcher. The script performs the following actions:

1. Sets the location of the Nuke executable for Nuke Bridge to use.
2. Launches Katana.

Windows and Linux sample launcher scripts for Katana containing the Nuke Bridge environment variable are available for download here.

To Configure a Katana Launcher Script for Windows:

You can use the template given below:

```
@echo off

rem ---------------------------------------
rem -------- KATANA Specific ------------
rem ---------------------------------------

set "KATANA_ROOT=C:\<KatanaInstallationDirectory>\Katana<version>"
set "PATH=%PATH%;%KATANA_ROOT%\bin"

rem ---------------------------------------
rem ----------- NUKE BRIDGE --------------
rem ---------------------------------------

set KATANA_NUKE_EXECUTABLE=C:\<NukeFolderLocation>\<NukeVersion>\<NukeApplication>.exe

rem -------- Start KATANA ------------
"%KATANA_ROOT%\bin\katanaBin.exe"
```

1. Edit the Katana installation location to include the version of Katana:

   ```
   set "KATANA_ROOT=C:\<KatanaInstallationDirectory>\Katana<version>"
   ```

   For example:

   set "KATANA_ROOT=C:\Program Files\Katana5.0v1”

2. Set and/or edit the Nuke executable location to include the file path of the custom Nuke launcher script:

   ```
   set KATANA_NUKE_
   EXECUTABLE=C:\<NukeFolderLocation>\<NukeVersion>\<NukeApplication>.exe
   ```

   For example:

   set KATANA_NUKE_EXECUTABLE=C:\Program Files\Nuke13.1v1\Nuke13.1.exe

Tip: If you are already using your own custom Katana launcher script, you can simply add this variable to your script and set the variable to the correct location.

To Configure a Katana Launcher Script for Linux

If you're using a script, you can copy and edit the template:

```
#!/bin/bash

export KATANA_NUKE_EXECUTABLE="/<nuke installation directory>/<version of nuke>/<nuke application>"

"/<katana installation directory>/<version of katana>/<katana application>"
```

You can set this variable in the terminal, or in your custom launcher script for Katana.

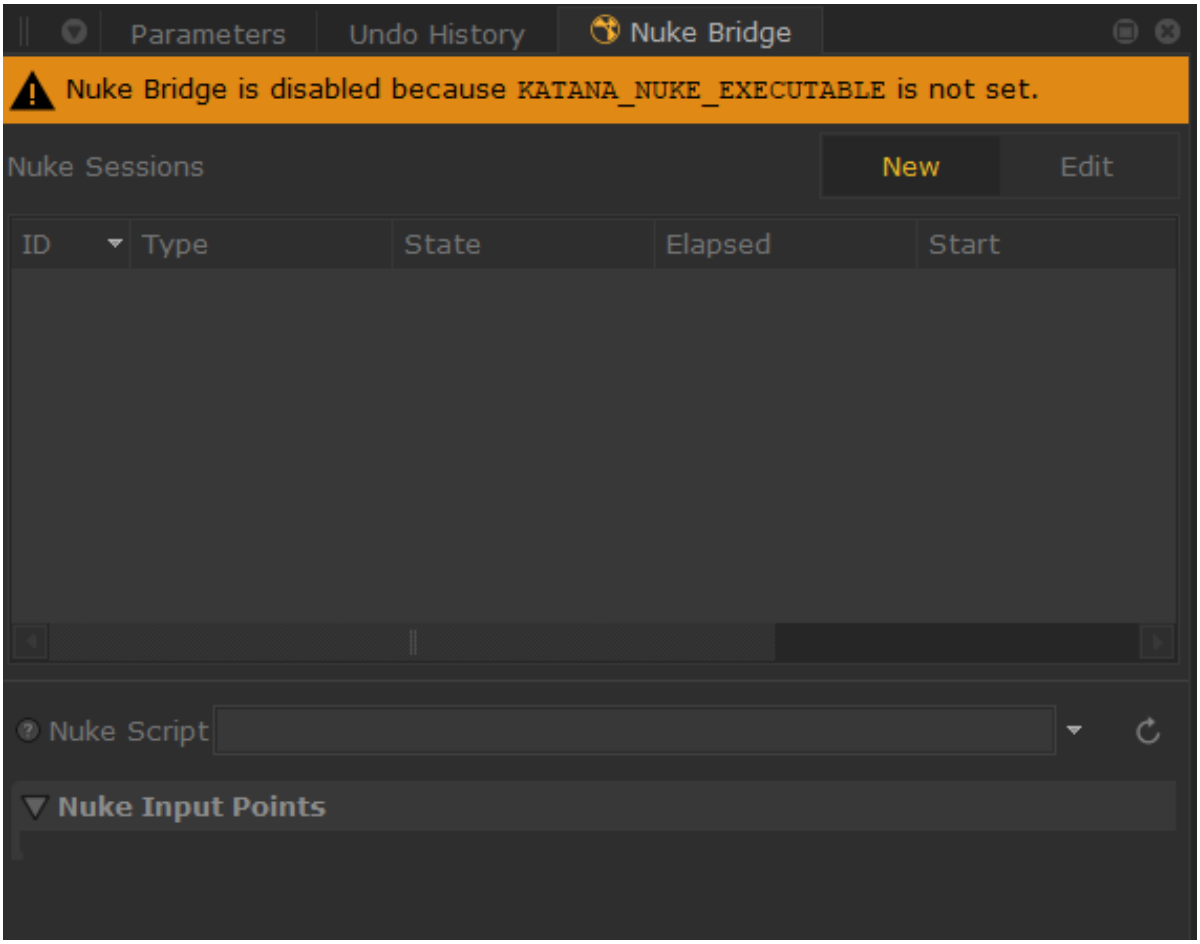
```
export KATANA_NUKE_EXECUTABLE="/<nuke installation directory>/<version of
nuke>/<nuke application>"
```

Add the file path of the Nuke executable file.

For example:

export KATANA_NUKE_EXECUTABLE="/opt/Foundry/Nuke13.1v1/Nuke13.1"

Note: If the KATANA_NUKE_EXECUTABLE variable in your Katana launcher script has been incorrectly set, a warning message at the top of the window will appear with a diagnostic.



If you're using a script, edit the path to the Katana executable.

Additional Steps for Setting up the Nuke Bridge With Only a nuke_i License

If you only have a nuke_i license, you can use all the feature of Nuke Bridge, with the following additional steps:

1. Create a custom launcher script for Nuke, in addition to the one that has been set up for standalone sessions.
2. Customize your Katana launcher script to reference this additional Nuke launcher script.

Creating an Additional Custom Launcher Script for nuke_i Licenses

Users will need to create a Nuke launcher script that contains an instruction to run the Nuke application with the correct flags.

- Windows: "C:\<PathToNuke>\<NukeVersion>\<NukeApplication>.exe" -i %*
- Linux: "/<path to nuke installation directory>/<version of nuke>/<nuke application>" -i "$@"

Setting a Custom Environment Variable for Katana's Launcher Script for nuke_i Licenses

The environment variable for set KATANA_NUKE_EXECUTABLE in Katana's launcher script will then need to point to the additional custom launcher created for Nuke, instead of the Nuke application.

- Windows: set KATANA_NUKE_EXECUTABLE=C:\<path to launcher>\nuke_launcher.bat
- Linux: export KATANA_NUKE_EXECUTABLE="<path to launcher>/nuke_launcher"

Launch Nuke and Prepare a Script for Nuke Bridge

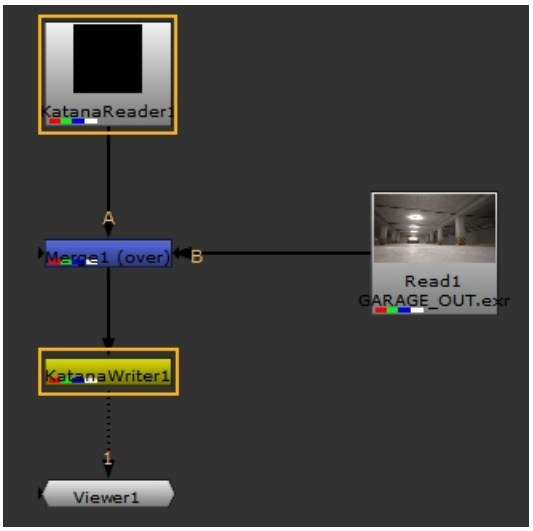
A Nuke script used by Nuke Bridge must contain at least one of the following nodes:

- KatanaReader - This node indicates to Nuke Bridge at which point the rendered image from Katana should be going into the Nuke script.
- KatanaWriter - This node indicates to Nuke Bridge at what point the composite should be fed back into Katana.

KatanaReader and KatanaWriter nodes work much like normal Read and Writer nodes but instead allow you to bring in renders from Katana and export them back out, and work with multiple renders and outputs at once.



Note: When performing an interactive comp, any KatanaWriter nodes in your Nuke Script must be connected to an active viewer node in Nuke in order for the comp to be streamed properly across both Katana and Nuke.



A node graph example for a simple composite of a backplate image set up for Nuke Bridge.

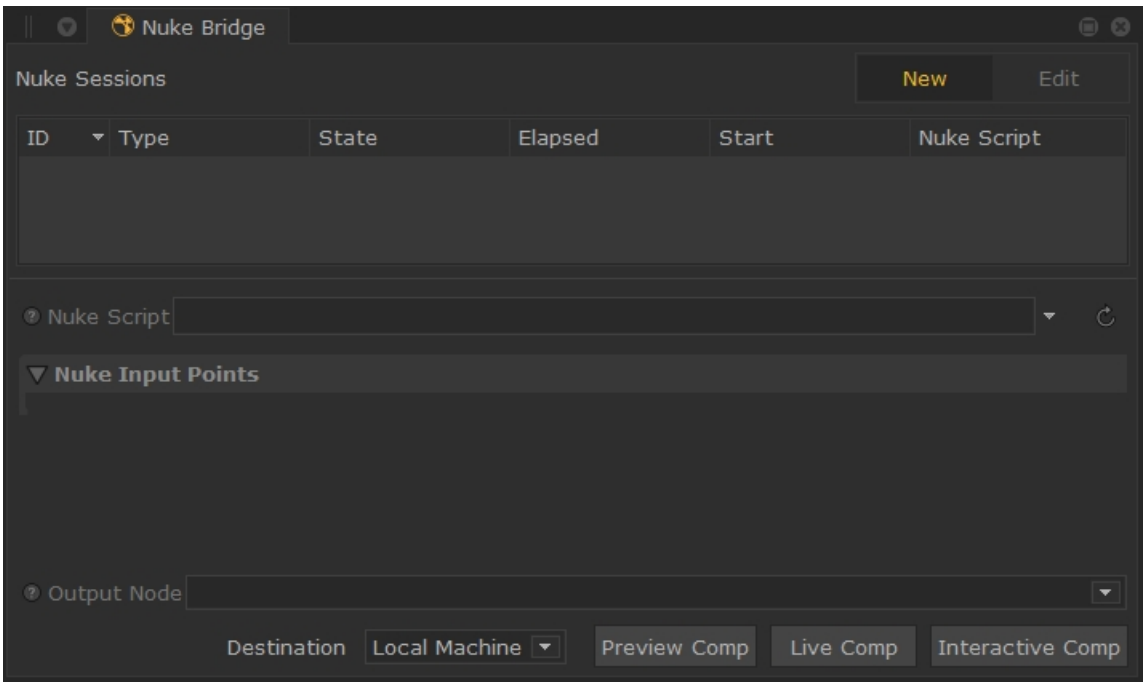
You can download the script of this node graph in the sample project.

Launch Katana and Test Nuke Bridge is Working

To test Nuke Bridge in Katana you can use our sample Katana scene or use your own project.

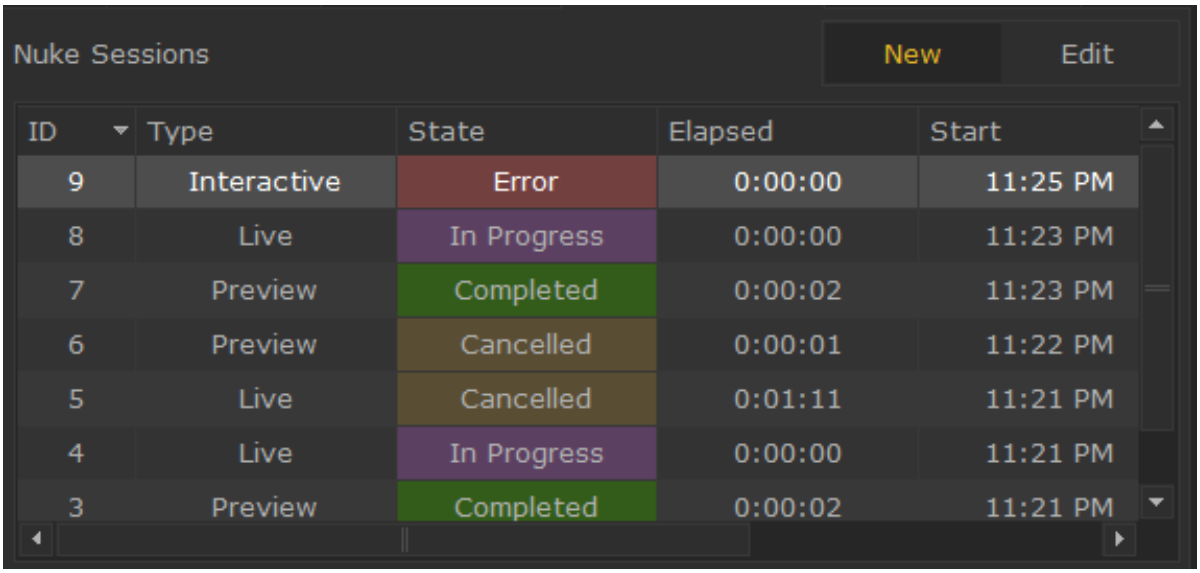
To test Nuke Bridge is operational:

1. Double-click on your Katana launcher script to run Katana.
2. In Katana, select **Tabs > Nuke Bridge** to open the Nuke Bridge window.



3. Specify a **Nuke Script** for Nuke Bridge to use, select a render in the **Nuke Input Points** section, the correct **Output Node** and **Destination** for your comp. Information on how to perform these steps can be found inUsing Nuke Bridge .
4. Perform a **Preview, Live** or **Interactive comp**.
5. If something in this process has been performed incorrectly, or the connection across your machine network has failed the **State** will change from **In Progress** to **Error.**

   You can see an example in the image below:



6. Additional diagnostic information on the failed session can then be accessed in the **Render Log** tab for troubleshooting.

Using Nuke Bridge

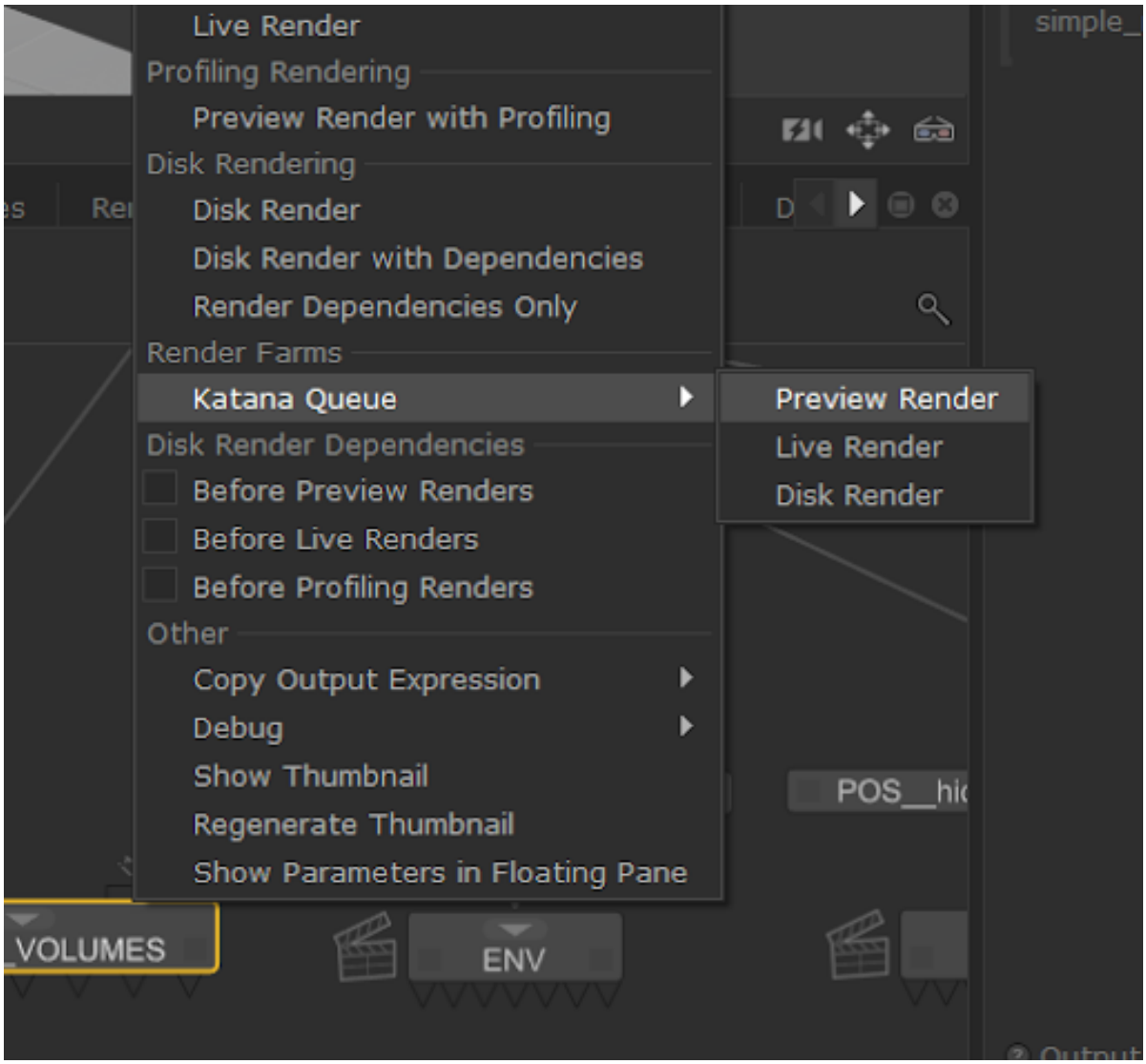
The workflow for using Nuke Bridge has the same initial steps for all three comp modes:

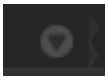
1. Create a render, or multiple simultaneous renders using the Katana Queue.
2. Choose a Nuke script in the Nuke Bridge panel to list the KatanaReader and KatanaWriter nodes.
3. Choose a render to send to a KatanaReader node.
4. Choose the KatanaWriter node to send the comp back to Katana.

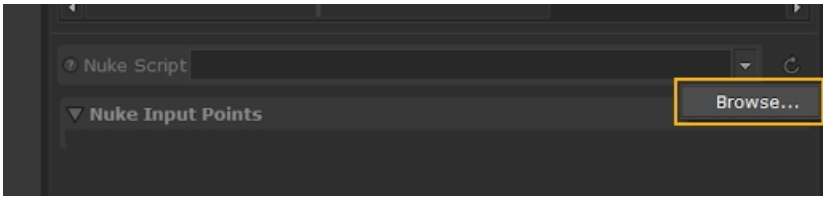
Once you've done this, you're ready to choose a comp mode to work with.

For detailed steps, follow these instructions:

1. Trigger a render by right-clicking the Render node in your scene, and selecting either **Preview Render**, **Live Render**, or **Disk Render**.

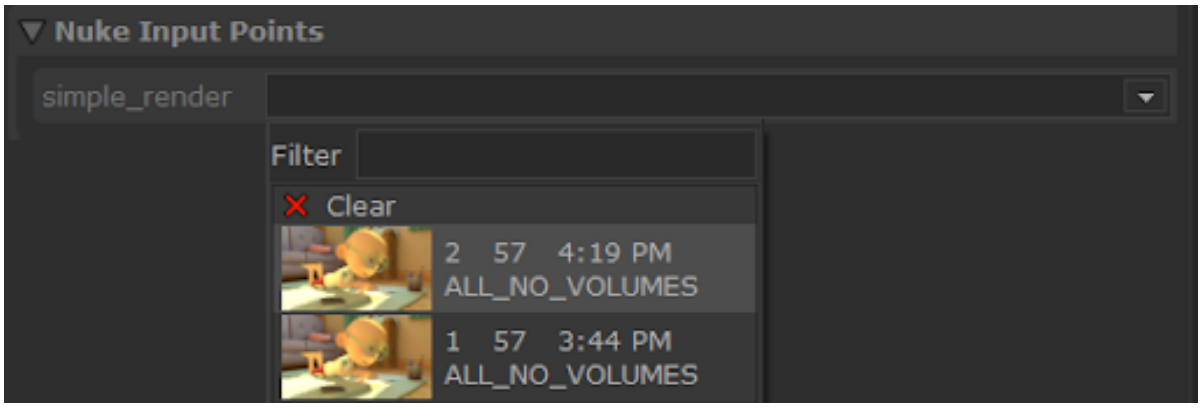


2. Open Nuke Bridge by going to **Tabs >Nuke Bridge**, selecting the  button in the section of Katana you want Nuke Bridge to open in.
3. Provide Nuke Bridge with a Nuke script by clicking the dropdown arrow and selecting **Browse.**

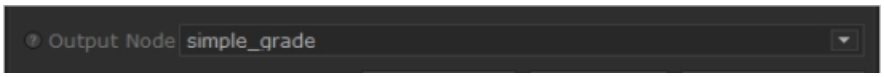


This opens an explorer window where you can find and import the Nuke script.

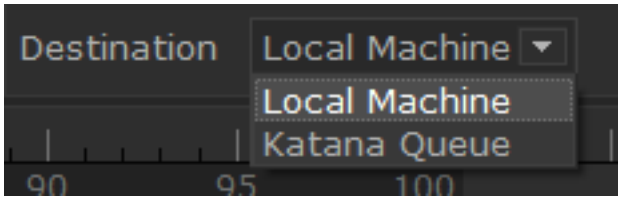
4. Once the Nuke script has been processed by Nuke Bridge, any KatanaReader nodes in the script then appear under the **Nuke Input Points** section.
Renders from Katana can then be fed into these input points through the dropdown menu.



5. Any KatanaWriter nodes in the Nuke script appear under the **Output Nodes** section.
   You can toggle between different KatanaWriter nodes for your output by using the dropdown menu.

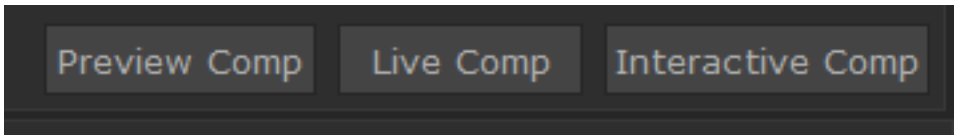


6. Set the destination of the comp to either your **Local Machine,** or the **Katana Queue.** This determines where the comp is performed.

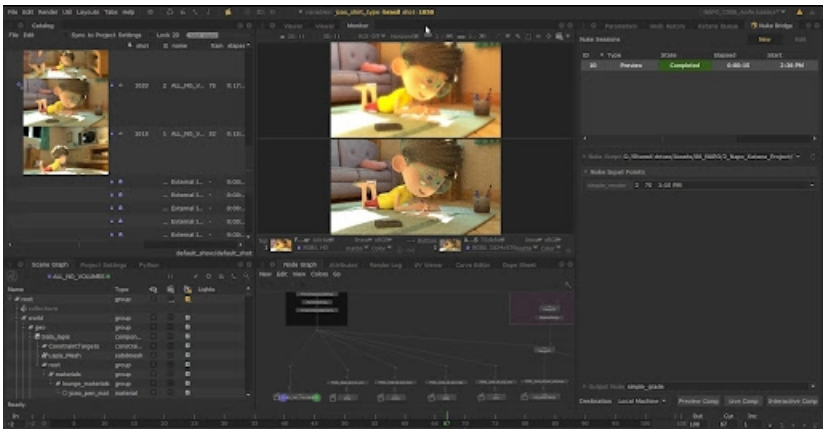


Note: Interactive Comps are unavailable when **Katana Queue** is selected as the destination.

7. Click **Preview Comp, Live Comp** or **Interactive Comp.**



8. The output of the comp is streamed to the **Monitor** tab in Katana and added to the catalog.

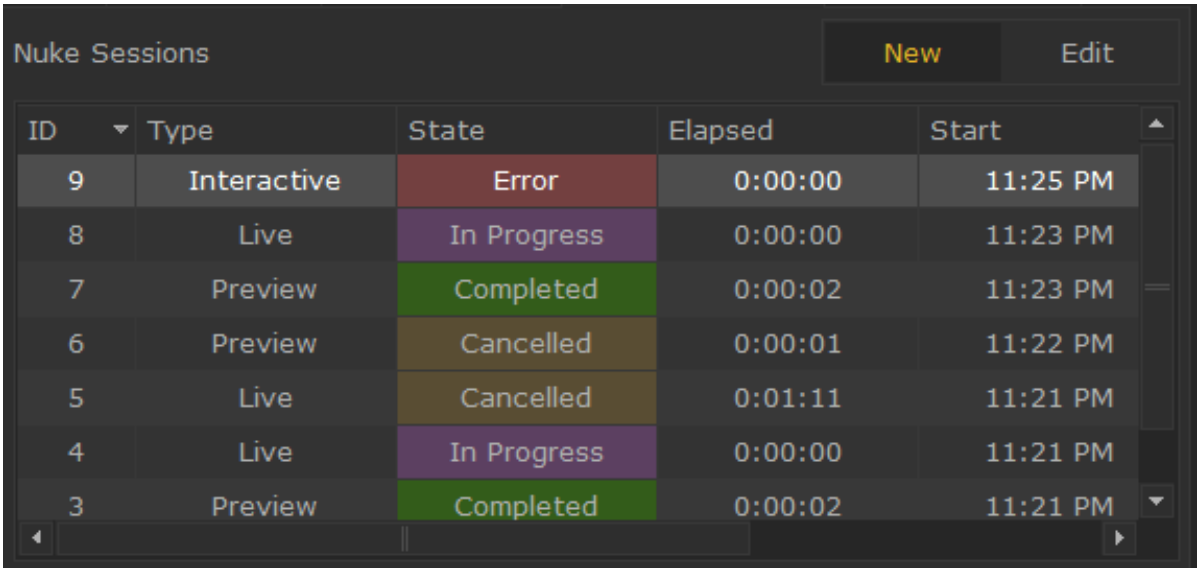


Note: When an Interactive Comp is launched, the output across Nuke's **Viewer** tab and Katana's **Monitor** tab are synced.

What's in the Nuke Bridge Panel?

The Nuke Bridge panel lets you start new comp modes and edit existing ones.

For example, you can see an example of Nuke Bridge's comp sessions running in the screenshot below:



- **Nuke Sessions** - Records details (ID, Status, Time, Nuke Script) of all comps started in the Nuke Bridge

- **ID -** An ID number assigned on the order in which the comp was performed. For example, the 9th comp performed in Nuke Bridge will be given the ID number 9.

- **Type -** The type of composite mode performed.
- **State -** Either **Completed**, **Cancelled**, **In Progress**, or **Error**.
- **Elapsed -** Runtime of the comp session.
- **Start** - When the comp session was performed.
- **Nuke Script** - The Nuke script used for the comp.
- **New -** Start a new comp in Nuke Bridge.This allows you to run multiple live or interactive comps simultaneously.
- **Edit -** Edit an existing 'in progress' comp.
- **Nuke Script** - Input the Nuke script for Nuke Bridge to perform a composite from the dropdown. The  button next to the dropdown allows you to reload the script in the event that changes have been made to it.
- **Nuke Input Points** - All KatanaReader nodes in the **Nuke Script**. Users map renders from Katana directly into these nodes.
- **Output Node** - All KatanaWriter nodes in the **Nuke Script**. These are used to view the output from Nuke.
- **Destination** - Determines where the output from the Nuke Bridge comp will be streamed to. Options are:
- **Local Machine** - Runs the comp and Nuke locally.
- **Katana Queue** - Runs the comp across a render farm, or machine network.

Preview Comp Mode: Trigger a Quick Preview of a Render from Nuke

Preview Comp mode is the most basic mode where Katana receives a snapshot of a render from a comp that's been made in Nuke.

To see any changes being made to your Katana scene, another render would need to be triggered, or another preview comp performed.

Nuke is run as a background process either locally, or on the farm.

You would use Preview Comp Mode if you want to get a quick and easy preview of your Katana render after a compositing process has been performed.



A render from Katana composited through Nuke Bridge with a preview comp.

Typical Workflow for Preview Comp Mode with Nuke Bridge

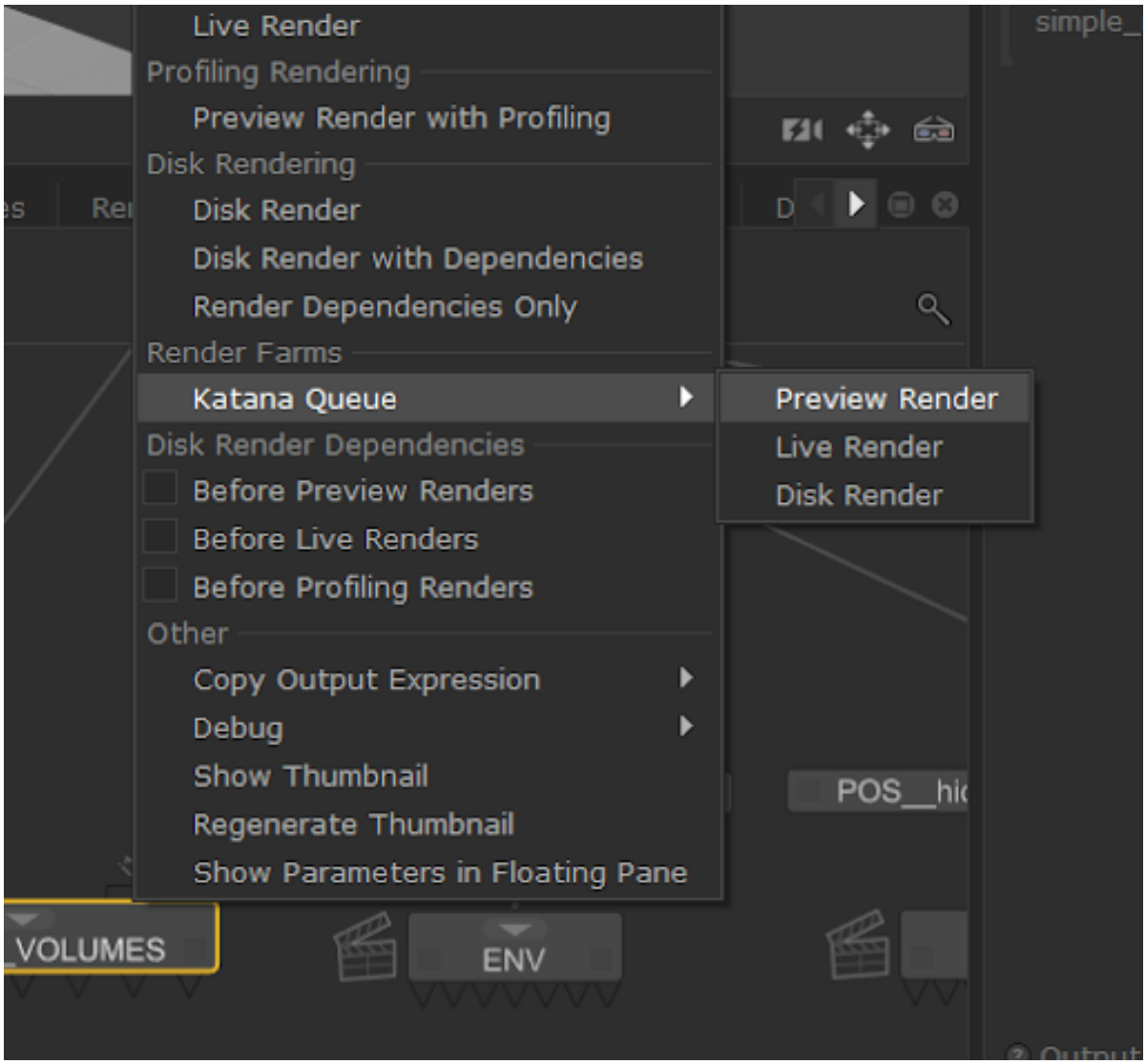
To use Preview Comp mode, Nuke is run in as a background process (headless) on either your local machine or the render farm. As this comp does not automatically update with changes from Katana, it is recommended you trigger either a preview or disk render of the scene to use with this mode.

In summary, preview comp mode has the following steps:

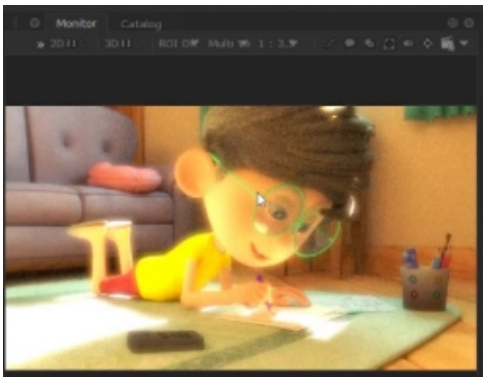
1. Set up Nuke Bridge according to the Set Up Nuke Bridge for Katana for Katana section.
2. Perform a preview, disk or live render.
3. **Destination** is either **Local Machine** or **Katana Queue**, depending on where Nuke is running.
4. Perform a **Preview Comp.**
5. Check the **Monitor** tab in Katana for the output of the preview comp. Details of the comp will also appear in the **Nuke Sessions** tab**.**

To see how to do this in Katana, follow these instructions:

1. With Nuke Bridge set up for Katana, perform a **Preview Render**, **Live Render**, or **Disk Render**.



2. Open Nuke Bridge, set up the comp with Inputs, Outputs, and a Destination. Then click the **Preview Comp** button.
3. The results from the composite are then streamed to your **Monitor** tab, registered as a **Catalog** entry, and the comp's metadata is recorded in the **Nuke Sessions** field.



4. Once the session is registered as complete, you can perform another Preview Comp to view new changes to the scene or a new render of the scene. To view changes that have been made to a Nuke script, users can use the Reload button  next to the **Nuke Script** dropdown.

Get Live Updates from a Nuke Process with Live Comp Mode

In Live Comp mode the results of the comp are streamed back to Katana as you make changes to your scene. Like Preview Comp mode, the results are streamed from a headless Nuke process that is using a Nuke script. However, unlike Preview Comp, renders or scene edits are updated automatically as you change them in Katana.

You would use Live Comp Mode to switch between different renders in a comp. What renders you choose to perform will depend on what the purposes of the Live Comp are. With Live Comp, users may switch between different renders under the **Input Points** section without needing to start a new comp. When **Live Comp** is used with live rendering, you can make changes to the properties of the scene such as lighting or graph state variables, and see those changes fed back through Nuke.

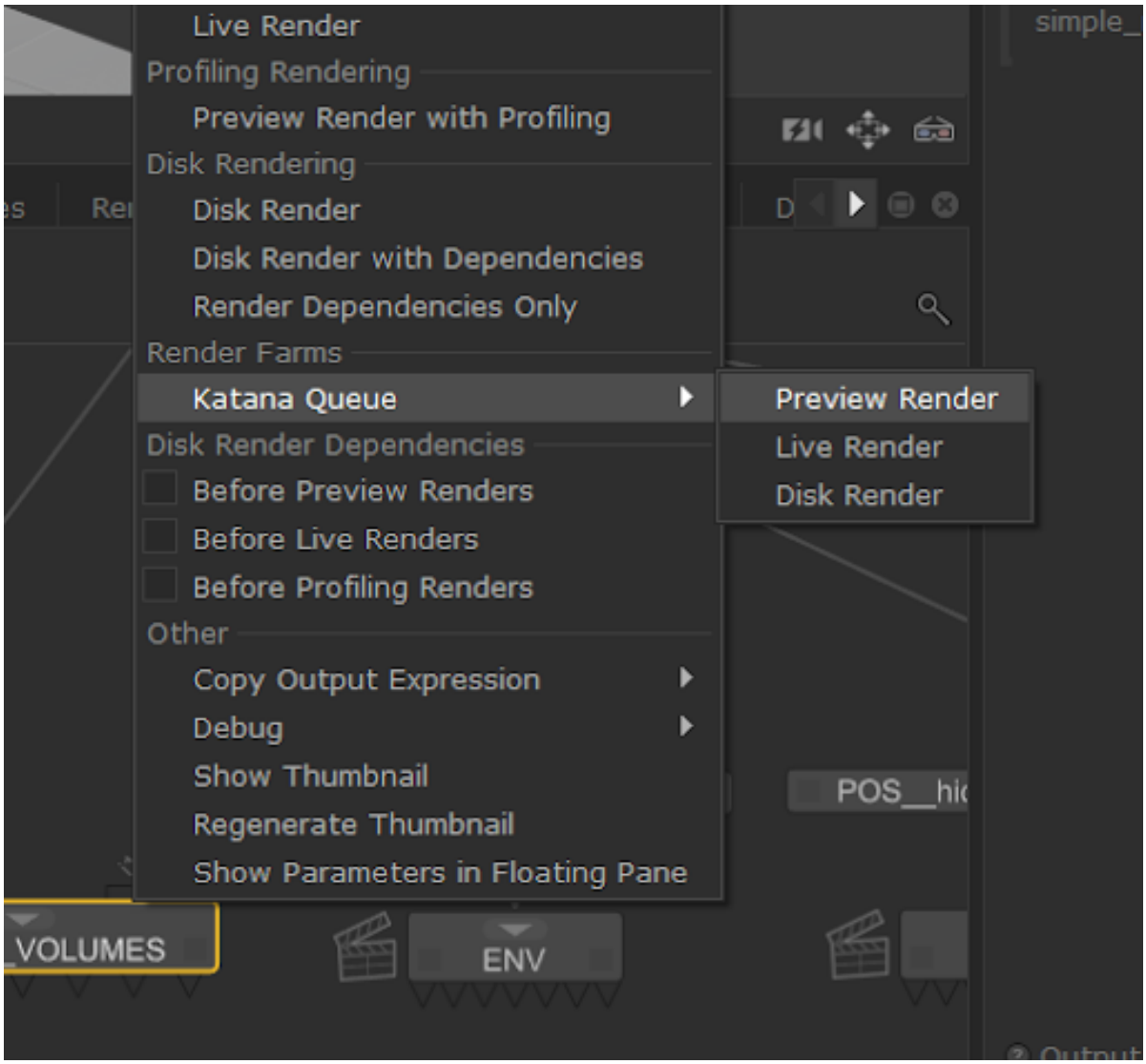
Typical Workflow for Live Comp Mode with Nuke Bridge

In summary, Live Comp mode has the following steps:

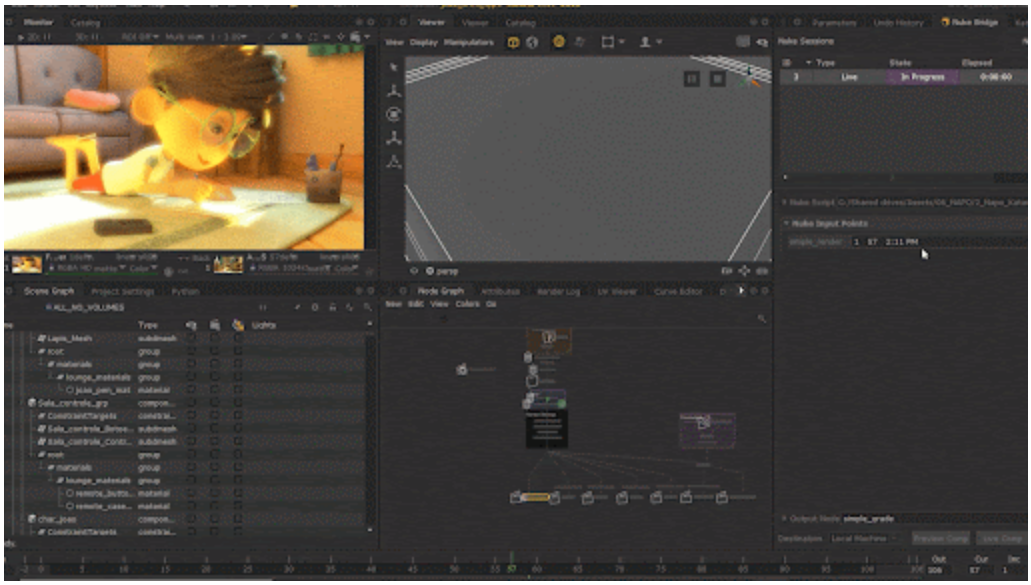
1. Set up Nuke Bridge according to the instructions in Set Up Nuke Bridge for Katana section.
2. Perform a preview, disk or live render. It is recommended that users use live renders with this feature, but preview or disk renders may also be used.
3. **Destination** is either **Local Machine** or **Katana Queue**, depending on where Nuke is running.
4. Perform a **Live Comp.**
5. Check the **Monitor** tab in Katana for the output of the live comp. Details of the comp will also appear in the **Nuke Sessions** tab**.**
6. Edit the details of your Katana scene to get live results fed back through Nuke.

To see how to do this in Katana in more detail, follow these steps:

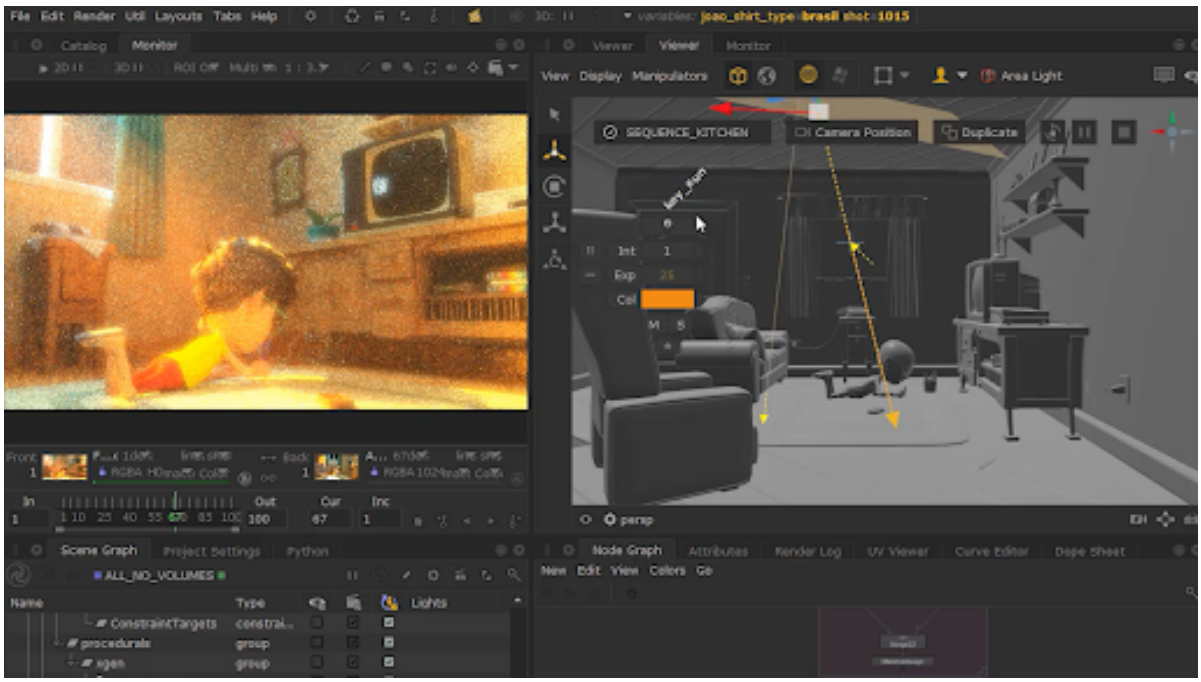
1. With Nuke Bridge set up for Katana, perform a **Preview Render**, **Live Render**, or **Disk Render**.

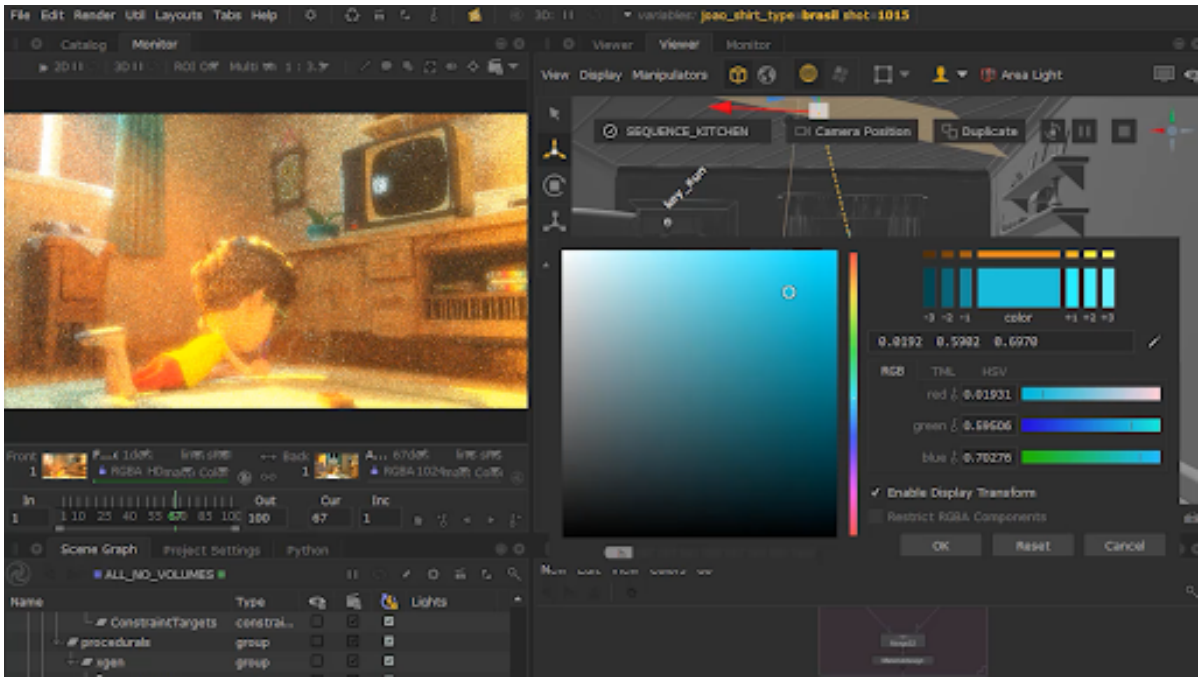


2. Open Nuke Bridge, set up the comp with Inputs, Outputs, and a Destination. Then click **Live Comp**.
3. The output of the Live Comp will appear in the **Monitor** tab, with a corresponding **Catalog** entry and **Nuke Sessions** entry. You can then switch between different renders in the **Input Points** dropdown without needing to start a new comp.

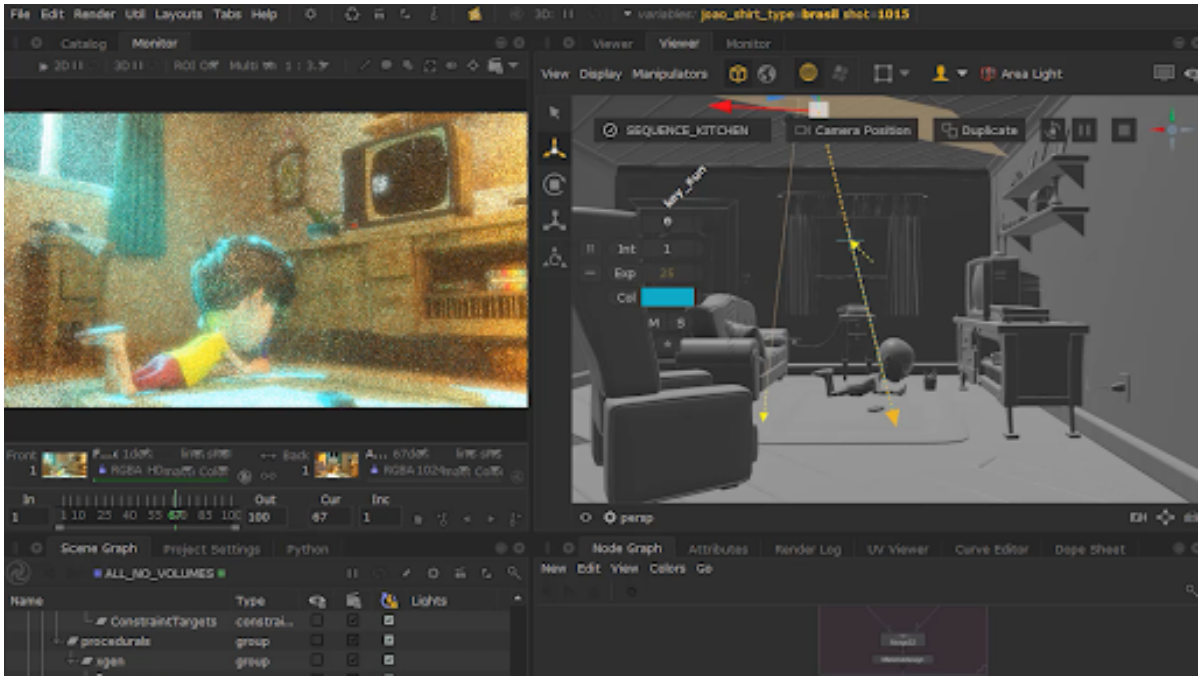


4. If you're using a live render to drive a **Live Comp**, you can make adjustments to the properties of the scene, such as lighting or graph state variables.





5. All updates from those changes are then communicated back to the **Monitor** tab through Nuke without needing to start a new comp, or a new render.



Interactive Mode: Run the Katana and Nuke Applications Together for Shared Edits

Interactive Comp Mode launches Nuke with the Nuke Script that was imported into Katana pre-loaded. You can then make edits in both your Nuke Script and Katana scene side by side, and receive corresponding updates in Katana, and vice versa.

You would use Interactive Comp Mode if you want to continue to make improvements in either application side-by-side. This provides a complete lighting and compositing environment for a single user.

Users in possession of only a nuke_i license can create a custom launcher script for Nuke as per the Nuke Licensing for Nuke Bridge section.

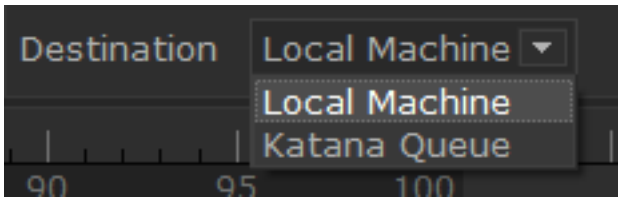
Typical Workflow for Interactive Comp Mode with Nuke Bridge

To use Interactive Comp mode, you need to have both Katana and Nuke running on your machine.

Your Nuke script must include KatanaReader and KatanaWriter nodes, and the KatanaWriter node must be connected to an active Viewer node.

In summary, interactive comp mode has the following steps:

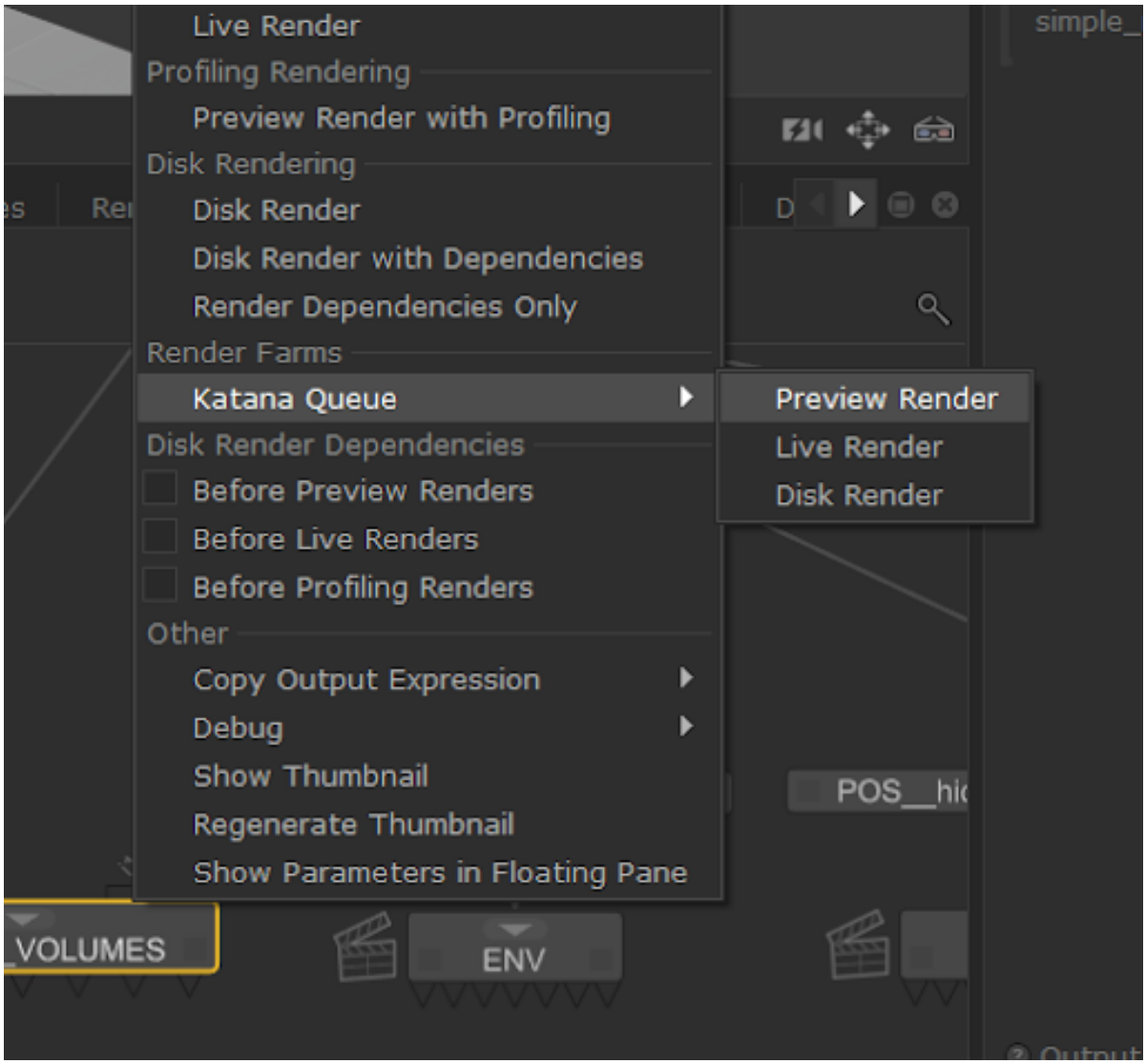
1. Set up Nuke Bridge according to the Set Up Nuke Bridge for Katana section.
2. Ensure the **Destination** is **Local Machine** and select **Interactive Comp**.



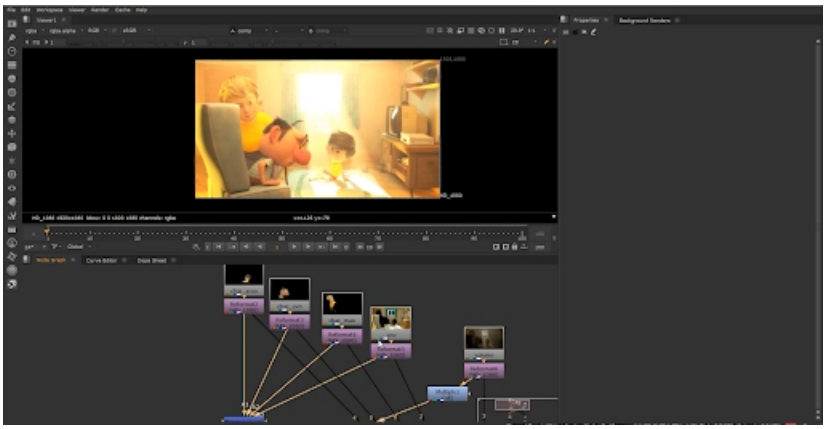
3. Edit both the Nuke script and Katana side by side.

To learn how to do this, follow these instructions:

1. With Nuke Bridge set up for Katana, perform a **Preview Render**, **Live Render**, or **Disk Render**.



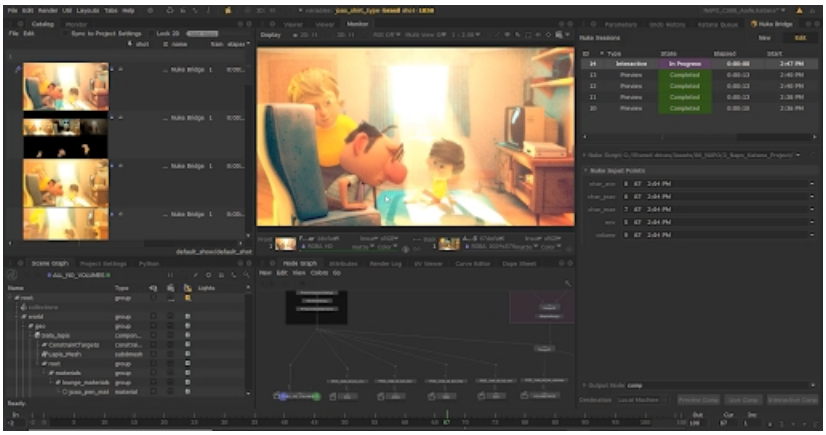
2. Open Nuke Bridge, set up the comp with Inputs, Outputs, and a Destination. Then click **Interactive Comp**.
3. Nuke then launches with the Nuke script that was specified to the **Nuke Script** dropdown in **Nuke Bridge.**



4. You can then make changes to your Nuke script. For example, perform color correction or grading.



5. Any changes made to the Nuke script are communicated back to the **Monitor** tab in Katana.



6. If you're using a Live Render to drive your Interactive Comp, you can also make edits to the properties of your Katana scene and have those changes fed back to Katana through Nuke, as with Live Comps.

Stopping a Nuke Bridge Comp

Once you're finished with Nuke Bridge comps, there are two ways in which they can be canceled:

1. Selecting the active Nuke Bridge comp you want to cancel, and hit the Esc key.
2. Ensure the Nuke Bridge comp you want to cancel is set as the **front buffer** in the **Catalog** tab**,** and go to **Render>Cancel Current Render.**

Additional Steps for Setting up the Nuke Bridge With Only a nuke_i License

Users in possession of only a nuke_r license will need to acquire a nuke_i license from Foundry to perform interactive comps.

If you only have a nuke_i license, you can use all the feature of Nuke Bridge, with the following additional steps:

1. Create a custom launcher script for Nuke, in addition to the one that has been set up for standalone sessions.
2. Customize your Katana launcher script to reference this additional Nuke launcher script.

Creating an Additional Custom Launcher Script for nuke_i Licenses

Users will need to create a Nuke launcher script that contains an instruction to run the Nuke application with the correct flags.

- Windows: "C:\Path\To\NukeVersion\NukeApplication.exe" -i %*
- Linux: "/opt/Foundry/Nuke13.1v1/Nuke13.1" -i "$@"

Setting a Custom Environment Variable for Katana's Launcher Script for nuke_i Licenses

The environment variable for set KATANA_NUKE_EXECUTABLE in Katana's launcher script will then need to point to the additional custom launcher created for Nuke, instead of the Nuke application.

- Windows: set KATANA_NUKE_EXECUTABLE=C:\path\to\additional\nuke_launcher.bat
- Linux: export KATANA_NUKE_EXECUTABLE="/path/to/additional/nuke_launcher"

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

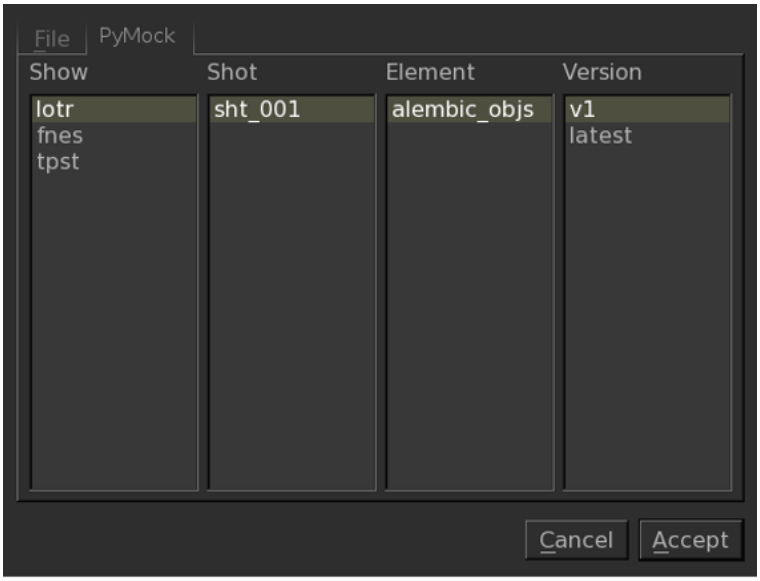
Katana includes an Asset API for plug-in authors, which consists of the following four core mechanisms:

- **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.



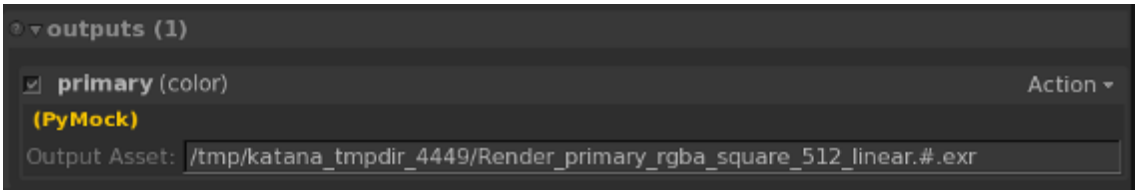
- **Parameter Display**

  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.



Asset Management System Plug-in API

The Katana Asset plug-in API is a Python and C++ interface for integrating Katana with asset management systems. It permits retrieval and publishing of assets within Katana. The asset management plug-in API provides four core mechanisms, which are described in Asset Management.

The Asset plug-in API does not provide any functions for traversing over a Katana scene graph or for editing nodes, and it is not a replacement asset management system. It is referenced when resolving a recipe and should therefore not traverse the **Node Graph** directly, or instantiate a scene graph iterator. An Asset plug-in is invoked during interactive Katana sessions and also during rendering.

Katana ships with an example Asset plug-in, called PyMockAsset. The source file **MockAsset.py** for the example plug-in is located in:

${KATANA_ROOT}/plugins/Src/Resources/Examples/AssetPlugins/

As well as source file **PyMockAssetWidgetDelegate.py** for the corresponding UI widget used with PyMockAsset, which is found in:

${KATANA_ROOT}/plugins/Src/Resources/Examples/UIPlugins/

PyMultiMockAsset is an extended version of PyMockAsset to allow a number of different asset resolving behaviors, such as publishing to a database or saving to a sandbox. This example uses assetIds with different prefix values to determine which behavior should be used. Further details are provided in the plug-in source file.



Note: Python-based AssetAPI plug-ins have been deprecated, and support for them will be removed. Moving forward, for performance and stability reasons, AssetAPI plug-ins should be written in C++.
For more information, refer to the support article Deprecation of Python-based AssetAPI plug-ins.

Concepts

Asset ID

An Asset ID is a serialization of an asset's fields. In a simple case, using the default **File** Asset plug-in, the Asset ID is the file path, but in more complex systems it could be an SQL query, a URL, a GUID or a direct string representation of the asset's fields, such as the PyMockAsset Asset ID shown below.

```
mock:///show/shot/name/version
```

As it's a single string, an Asset ID can be passed as part of an argument string to a sub-process, such as a shell command or a procedural. It is important therefore that the format of an Asset ID is such that it can be easily found in a larger string and parsed.

Asset Fields

The fields of an asset are the key components needed to retrieve an asset from an asset management system. Katana assumes that an asset has a **name** field and - if provided - also uses a **version** field.

Asset Attributes

An asset can optionally have attributes where additional metadata is stored, such as comments, or information about the type of asset.

Katana does not rely on particular attributes to exist, but it presumes that there is a mechanism in place for this additional data to be read from and written to.



Note: It is fine to leave these methods unimplemented if your asset management system has no use for them.

Asset Publishing

Assets are published by users. When an asset is published it is in a finalized state, accessible to other users. Publishing can involve incrementing the asset version.



Note: Any change that alters the project's **katanaSceneName** whilst saving a scene triggers a call to **SyncAllOutputPorts()**. This ensures render outputs affected by this change are correct.

Transactions

A Transaction is a container for submitting multiple publish operations at once. Rather than submit one publish operation per asset, operations can be grouped. This means that if an error occurs whilst publishing many assets, the whole operation may be aborted.

```
beginTransaction (createTransaction)
publish asset A
publish asset B
publish asset C
```

```
endTransaction (commit)
```

The transaction is final only after the **endTransaction(commit)** operation.

A transaction must have a **commit** method and a **cancel** method. The **cancel** method can be used to rollback.



Note: Implementing plug-in support for Transactions is optional.

Creating an Asset Plug-in

A Python Asset plug-in is created by making a new Python file in an **AssetPlugins** sub-directory of a folder in a **KATANA_RESOURCES** directory.



Note: Asset management plug-ins can also be written in C++. See The C++ API for more on this.

Core Methods

The core methods for an Asset plug-in are:

Handling Asset IDs

- **buildAssetId()**

  Serialize asset fields into an Asset ID.
- **getAssetFields()**

  Deserialize an Asset ID into asset fields.
- **isAssetId()**

  Check if a string is an Asset ID.

Publishing an Asset

- **createAssetAndPath()**

  Create an entry for a new asset and optionally pre-publish it. This could have very little in it if your asset management system does most of its work post creation in postCreateAsset.

- **postCreateAsset()**

  Publish the new asset. This could have very little in it if your asset management system does most of its work immediately when the resource is allocated in createAssetAndPath.

Retrieving an Asset

- **resolveAsset()**

  Convert an Asset ID to a file path.

- **resolvePath()**

  Convert an Asset ID and a frame number to a file path.



Note: Python-based AssetAPI plug-ins have been deprecated, and support for them will be removed. Moving forward, for performance and stability reasons, AssetAPI plug-ins should be written in C++.
For more information, refer to the support article Deprecation of Python-based AssetAPI plug-ins.

Publishing an Asset

The methods for publishing an Asset in a custom Asset Management System are **createAssetAndPath()** and **postCreateAsset()**.

createAssetAndPath() creates or updates an asset entry, given a collection of fields and an asset type. It returns the ID of the asset, which resolves to a valid file path. It is invoked prior to writing an asset. The fields passed to **createAssetAndPath()** may be the result of a decomposed Asset ID stored as a parameter on a node.

Both **createAssetAndPath()** and **postCreateAsset()** are used by Katana mechanisms that write assets. The Asset ID returned from **createAssetAndPath()** is used to create the fields passed to **postCreateAsset()**. The result from **postCreateAsset()** is used from that point onward (such as in the **File** > **Open Recent** menu or in any references to that asset ID in the current scene):

```
assetFields1 = assetPlugin.getAssetFields(assetId, True)
id1 = assetPlugin.createAssetAndPath(..., assetFields1, ...)
[Write Katana project file, for example]
assetFields2 = assetPlugin.getAssetFields(id1, True)
id2 = assetPlugin.postCreateAsset(..., assetFields2, ...)
```

This is done to allow a temporary file path to be used for the write operation. The LookFileBake node and the Render node use these methods.

createAssetAndPath()

The arguments for **createAssetAndPath()** are:

- **txn**

  The Asset Transaction (implementation optional). Can be used to group create/publish operations together into one cancelable operation. This transaction is created via the createTransaction method.

- **assetType**

  A string representing which of the supported asset types is published. See Asset Types and Contexts for a list of the asset types, and contexts.

- **fields**

  A dictionary of strings containing the asset fields that represent the location of the asset. These are typically produced by de-serializing an Asset ID stored as a parameter on a node (such as a LookFileBake node). These fields are based on the Asset ID returned by **createAssetAndPath()**.

- **args**

  A dictionary containing additional information about what asset type to create. For example, should we increment the asset version? Is it an image, is it a Katana file? This is populated directly by the caller of **createAssetAndPath()** and varies with the asset type.

- **createDirectory**

  A Boolean indicating that a new directory for the asset needs to be created.

createAssetAndPath() should return the Asset ID of the newly created asset. This may be different to the serialized Asset ID representation of the fields passed in. For example, if **createAssetAndPath()** were to **versionUp** the asset the returned Asset ID would likely be different to the serialized fields passed in. The returned Asset ID can be stored as a parameter on the node using this plug-in (if it is being used by a node).

The important arguments are **assetType**, **fields** and **args**. There are no rules for how the args dictionary is populated. It depends on the calling context and the Asset Type that **createAssetAndPath()** was invoked for.

postCreateAsset()

postCreateAsset() is invoked after Katana has finished writing to an asset file and is used to finalize the publication of the asset.

The args dictionary for this type contains:

- **txn**

The Asset Transaction.

- **assetType**

  A string representing which of the supported asset types is published. See Asset Types and Contexts for a list of the asset types, and contexts.

- **fields**

  The fields that represent the location of the asset. These fields are the identical to those given to **createAssetAndPath()**.

- **args**

  A dictionary of strings containing additional information about what asset type to create. For example, should we increment the asset version? If it is an image, what resolution should it be?

Examples

Selecting **File** > **Version Up and Save** triggers **createAssetAndPath()** to be invoked with an **args** dictionary, in which the **versionUp** and **publish** keys are set to 'True'. This results in a different Asset ID to that of the serialized fields passed in. versionUp indicates that a new version of the asset should be published.

Selecting **File** > **Save** triggers **createAssetAndPath()**, invoked with versionUp and publish set to **False**, unless a custom asset browser has been written. In that case, versionUp and publish are based on the values returned from the **getExtraOptions()** method of a custom browser class. See Configuring the Asset Browser for more on this.

Asset Types and Contexts

The following asset types are available to the **AssetAPI** module:

- **Katana project**

  kAssetTypeKatanaScene

- **Macro**

  kAssetTypeMacro

- **Live Group**

  kAssetTypeLiveGroup

- **Image**

  kAssetTypeImage

- **Look File**

kAssetTypeLookFile

- **Look File Manager Settings**

kAssetTypeLookFileMgrSettings

- **Alembic Files**

kAssetTypeAlembic

- **Casting Sheets**

kAssetTypeCastingSheet

- **Attribute Files**

kAssetTypeAttributeFile

- **F Curves**

kAssetTypeFCurveFile

- **Gaffer Light Rig**

kAssetTypeGafferRig

- **Scene Graph Bookmarks**

kAssetTypeScenegraphBookmarks

- **Shaders**

kAssetTypeShader

In addition, the following list of contexts is available inside the **AssetAPI** module, and passed as hints to the asset browser whenever it is invoked:

- kAssetContextKatanaScene.
- kAssetContextMacro.
- kAssetContextLiveGroup.
- kAssetContextImage.
- kAssetContextLookFile.
- kAssetContextLookFileMgrSettings.
- kAssetContextAlembic.
- kAssetContextCastingSheet.
- kAssetContextAttributeFile.
- kAssetContextFCurveFile.
- kAssetContextGafferRig.
- kAssetContextScenegraphBookmarks.
- kAssetContextShader.

- kAssetContextCatalog.
- kAssetContextFarm.

A constant to hold the relationship between assets has been added. This constant is used when the **getRelatedAssetId()** function is called:

- kAssetRelationArgsFile.

Accessing an Asset

The **resolveAsset()** method must be implemented in order for Katana to gain access to the asset itself.

It takes an Asset ID as its first argument and returns a string containing a file path to the asset. This handle is a path to a file that can be read from and written to.



Note: An Asset plug-in must not attempt to use any **NodegraphAPI**, **user interface**, or **callback** modules when resolving an Asset ID. This is because Asset ID resolution occurs at render time, when these modules are not available. Reading from the scene graph while writing to it results in undefined behavior.

Additional Methods

In addition to the core methods that need to be implemented by an Asset plug-in there are additional methods, many of which are variants.

reset()

Triggered when the user requests that Katana flush its caches. This is used to clear local Asset ID caches to allow retrieval of the latest version of an asset.

resolveAllAssets()

Used for expanding Asset IDs in a string containing a mix of Asset IDs and arbitrary tokens, such as a command. It takes a single string parameter, which may contain one or more Asset IDs, and replaces them with resolved file paths. **resolveAllAssets()** is used by:

- Python expressions, which have access to a function called **assetResolve()** that resolves a string of Asset IDs split by white space.
- String parameters, which has a method called **getFileSequenceValue()** that returns the value of the string with automatic expansion of Asset IDs into file paths.
- ImageWrite node postScripts. An ImageWrite node can execute post scripts commands. The Asset IDs in these commands are automatically expanded.

resolvePath()

This resolves an Asset ID and frame number pair, where time is a factor in determining the asset resolution (such as a sequence of images). **resolvePath()** is called in place of **resolveAsset()** whenever time is a significant factor in asset resolution.

resolvePath() is used extensively for resolving procedural arguments in render plug-ins. It is used by the Material and RiProcedural resolvers, and the Look File Manager. It can be accessed in Attribute Scripts via the **AssetResolve()** function in an **Attribute Script Util** module.

resolveAssetVersion()

This accepts an Asset ID that references a tag or meta version such as **latest** or **lighting** and returns the version number that it corresponds to. It also accepts an Asset ID that contains no version information and an optional **versionTag** parameter, and produces the version number that corresponds to the **versionTag** argument.

This is used by the LookFile resolver, Katana in Batch mode, the Casting Sheet plug-in, and the Importomatic user interface.

createTransaction()

It allows Katana to create assets in bulk. If createTransaction is implemented to return a custom transaction object, then the object must have **commit** and **cancel** methods that take no arguments. The **commit** method should submit the operations accumulated in the transaction to the Asset Repository. The **cancel** method should rollback the publish operations accumulated in the transaction.

The transaction is passed by Katana to **createAssetAndPath()** and to **postCreateAsset()**. An example of this is in the Render node.



Note: For Python asset plug-ins, this method must be implemented but it may return **None**. In C++, this method may be implemented; where it is, it must return an asset transaction object.

containsAssetId()

Reports if a string contains an Asset ID.

The string parameter uses this method prior to expanding the Asset IDs it may contain, when **getFileSequenceValue()** is called.

getAssetDisplayName()

Is used to produce a short name for an asset. For example, a name that can be used in the UI.

This is used by the Alembic Importomatic plug-in and the LookFileManager.

getAssetVersions()

Lists the versions that are available for an asset as a sequence of strings.

This is used by the Importomatic, to allow users to choose an asset version in the Importomatic versions column and by the CastingSheet plug-in.

getUniqueScenegraphLocationFromAssetId()

Provides a scene graph path for an asset, as a string, so that it can be placed easily in the **Scene Graph** tab, and is currently used by the LookFileManager.

getRelatedAssetId()

Given an Asset ID and a string representing a relationship or connection, returns another Asset ID. For example, with a shader file that has an Args file **getRelatedAssetId()** can be used to get the Asset ID of the Args file from the Asset ID of the shader. The contexts listed in Asset Types and Contexts are passed to **getRelatedAssetId()**.



Note: If **getRelatedAssetId()** returns either **None**, or an empty string, Katana looks up the Args file in the default fashion, relative to the **.so** file location.



Note: If **getRelatedAssetId()** returns anything other than **None** or an empty string, Katana attempts to load the returned Asset ID. If, for any reason, that Asset ID is not valid, Katana does not fall back to the default behavior, but gives a load error.

getAssetIdForScope()

This truncates an Asset ID to the given scope, where the scope is an asset field.

For example:
```
getAssetIdForScope( "mock://myShow/myShot/myName/myVersion", "shot" )
```

Produces:
```
mock://myShow/myShot
```

The returned Asset ID no longer contains the **name** and **version** components.

This is used by the **assetAttr()** built-in function that Python expressions have access to, and by Katana internally.

setAssetAttributes()

Allows users to set additional metadata on an asset.

This is not used by anything in the Katana codebase. It is entirely up to the users to make use of this function.

getAssetAttributes()

Allows users to store additional metadata on an asset.

The casting sheet example plug-in uses this method and Python expressions have access to an assetAttr built-in method that retrieves asset attribute information.

Top Level Asset API Functions

The top level Asset API functions can be found by opening a **Python** tab and typing:
```
help( AssetAPI )
```

The most useful are:

- **SetDefaultAssetPluginName()**

  Sets the default asset plug-in to use in the user interface for this Katana project.
- **GetDefaultAssetPlugin()**

  Retrieves an asset plug-in by name.
- **GetAssetPluginNames()**

  Lists the names of all the currently registered asset plug-ins.

LiveGroup Asset Functions

A studio may decide to use permissions for working with certain assets on a project. These permissions may depend on the name of the current user, the name of the user's workstation, or certain environment variables for a project, such as show, shot, or sequence. Katana's AssetAPI supports such access permissions through a dedicated function, **checkPermissions()**, which is called for certain LiveGroup operations. When a function to check permissions in a specific context is called, the asset API plug-in queries the Asset Management System (AMS) to check general permissions or permissions for working with the asset with the given ID in the given context. Checking permissions for a given ID can be used to check whether the current user has sufficient permissions to edit the asset or whether the asset has already been checked out for editing.



Note: It is possible, with a custom implementation leveraging the Asset Management System, to inform users of editable permission errors, such as when another user is currently editing the LiveGroup source of the node you're attempting to edit. If another LiveGroup node references the same LiveGroup **source** and has been made editable by another user, an error is displayed and the state of the node is not changed.

The function signature for checking permissions is:

```
checkPermissions(assetID: string, context: map of string to string): bool
```

The context dictionary contains information about the context from which to check permissions, with names of information fields as keys and values of information fields as values. For example, the following might be produced:

- action = edit
- shot = ts520
- show = srow
- username = name
- workstation = seat

When the function to run a custom asset plug-in command is called the asset API plug-in uses the AMS to check if the command succeeds or fails. The function signature to run the plug-in command is:

runAssetPluginCommand(assetID: string, command: string, commandArgs: map of string): bool

The command parameter receives the command to execute, for example:

- acquireLock
- releaseLock

The *commandArgs* dictionary contains information about the arguments with which to customize the execution of the given command, with names of command arguments as keys and values of command arguments as values. The *commandArgs* dictionary may be empty.

Extending the User Interface with Asset Widget Delegate

Katana provides a mechanism for configuring the asset related parts of its user interface. This is achieved by implementing and registering an AssetWidgetDelegate.

The PyMockAssetWidgetDelegate.py provides a good reference. This file is shipped with Katana in:

${KATANA_ROOT}/plugins/Src/Resources/Examples/UIPlugins/

This allows users to:

- Configure the asset/file browser. Typically this is done by extending with a **custom asset browser** tab.
- Implement a custom Python QT widget for displaying and editing Asset IDs in the **Parameters** tab.
- Implement a custom Python QT widget for displaying and editing render output locations in the **Parameters** tab.
- Customize the QuickLink paths used by the file browser.

To create an AssetWidgetDelegate plug-in, create a new Python file and place it in a directory called **UIPlugins** in a folder in your **KATANA_RESOURCES**.



Note: The **UI4** module is the main **Python** module for working with the Katana user interface.

Configuring the Asset Browser

The entry point for extending the Katana asset browser is the method **configureAssetBrowser()**, which must be implemented in your AssetWidgetDelegate plug-in. **configureAssetBrowser()** takes a single browser argument, which is the Katana Asset Browser to configure. At its core the **Asset Browser** is a Qt dialog window (QDialog) with additional utility methods. The most useful of these are:

- **addBrowserTab()**
  Add a new tab to the Asset Browser.



Note: The custom browser tab added using **addBrowserTab()** should emit a **selectionValidSignal** signal to indicate a change in selection validity and therefore the state of the Asset Browser **Accept** button, for example:
`browserTab.selectionValidSignal.emit(browserTab.selectionValid())`
The browser dialog listens for this signal from the currently viewed tab and sets the enabled state of its **Accept** button accordingly.

- **addFileBrowserTab()**
  Add a standard file browser tab to the Asset Browser.
- **getCurrentIndex()**
  Return the index of the currently open tab.
- **setCurrentIndex()**
  Set the currently open tab.

The base implementation of **configureAssetBrowser()** sets the window title from the given hints and creates a file browser tab. If you want to avoid creating a **file browse**r tab, implement a **shouldAddFileTabToAssetBrowser()** method with a return value of False.

The following methods exist but need minimal implementation:

- **setSaveMode()**
  Tells us whether the browser is invoked for opening a file or for saving one. If **saveMode** is **True**, then the browser has been opened for saving a file.
- **selectionValid()**
  Checks whether the current asset path refers to a valid asset. For a **file browser** dialog window this returns false if a chosen path does not exist.
- **setLocation()**
  Sets the default location with which to open the browser.

• **getExtraOptions()**
This is used to support a **versionUp** and a **publish** option for LookFileBake and create a new Katana file. If those options are displayed in the custom user interface Katana retrieves them using this method:

```
{"versionUp" : "False" / "True", "publish" : "False" / "True" }
```

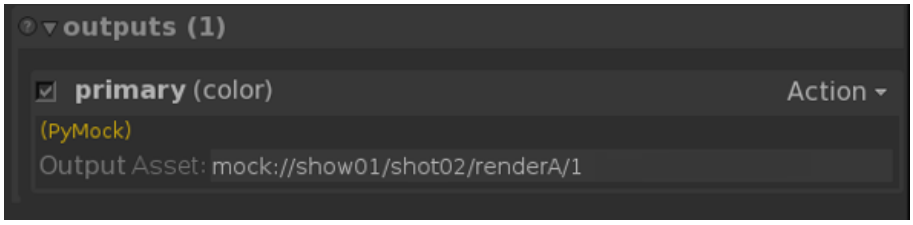


Note: The function **getExtraOptions()** should return a dict.

The Asset Control Widget

The AssetWidgetDelegate plug-in API makes it possible to replace the default string widget that allows users to view and edit an Asset ID in the node **Parameters** tab.

Typically you edit the fields of an asset through a UI. Internally those fields are serialized into a single string as an Asset ID, and stored as a parameter on a node.



Using a custom Asset Control Widget you can replace the widget displaying the fields. Katana knows to use the custom widget through the **assetIdInput hint**, which is associated with all string parameters that represent an Asset ID.

Implementing A Custom Asset Control Widget

The entry point that Katana needs, in order to create a custom asset control widget is the **createAssetControlWidget()** method of our custom AssetWidgetDelegate class.

The **createAssetControlWidget()** method instantiates the SimpleMockAssetControlWidget, which must inherit from BaseAssetControlWidget. BaseAssetControlWidget is a QT QWidget with an HBoxLayout. **createAssetControlWidget()** then adds the control widget to the parent layout. The parent is a QWidget and part of the **Parameter** tab.

The following methods must be implemented by an asset control widget:

- **buildWidgets()**

  This is invoked by the BaseAssetControlWidget constructor to build the child widgets. This is where most of the work happens.

- **setValue()**

  Updates this widget with the given Asset ID.

- **getValue()**

  Return the Asset ID from this widget.

- **setReadOnly()**

  Enable/Disable editing of this widget.

The BaseAssetControlWidget supplies an **emitValueChanged()** method for notifying Katana that the user has changed the Asset ID in the widget. This must be called when the value in the UI has changed.

Asset Render Widget

The Asset Widget Delegate allows customization of the display of the render output location shown in a Render node's **Parameters** tab. This is useful for when rendering to a location in a custom asset management system.

This output location could be an automatically generated temporary path or one set explicitly using a Render Output Define node. It is set on a Render Node and therefore the Asset Render Widget is read-only.

Implementing an Asset Render Widget

Implementing an Asset Render Widget is optional. The Asset Management user interface does not require this. The entry point for a custom widget delegate is similar to that of the Asset Control Widget.

The Asset Widget Delegate must implement **createAssetRenderWidget()**, which in turn must return a class that inherits from **baseAssetWidgetDelegate()** and implements two methods, **buildWidgets()** and **updateWidgets()**.

createAssetRenderWidget() has an additional **outputInfo** argument, which is a dictionary of output information and must be passed to the **BaseAssetRenderWidget** constructor. The outputInfo dictionary contains the output location's Asset ID along with additional information (such as the image file type and resolution). **BaseAssetRenderWidget** provides a utility method, **getOutputInfo()** for accessing this dictionary.

The key for the Asset ID of the output location is **outputLocation**.

Additional Asset Widget Delegate Methods

There are several methods used to make small customizations to the Asset Management UI. These are implemented as overridable methods on the Asset Widget Delegate.

addAssetFromWidgetMenuItems()

Allows you to extend the menu item to the right of an Asset ID in the **Parameters** tab with additional items.

```
def addAssetFormWidgetMenuItems(self, menu):
       menu.addAction("Custom Action",
                           self.__customCallback)

def __customCallback(self, *args):
       print args
```

shouldAddStandardMenuItem()

When **False** is returned from this method, the menu item to the right of an Asset ID in the **Parameters** tab is not displayed when clicked on.

shouldAddFileTabToAssetBrowser()

The **File** tab is not displayed in the Asset Browser when this is set to return **False**.

getQuickLinkPathsForContext()

For customization of the quicklink paths displayed at the bottom of the **File** tab. Must return a sequence of file paths.

Locking Asset Versions Prior to Rendering

In many pipelines it is considered desirable to lock all the assets used in a shot to specific versions prior to rendering. When an asset is locked, meta versions (or tags) are resolved to a fixed static version, represented by a number. This ensures that the same asset version is used for rendering all frames. Conventional ways of doing this include creating a look-up table to specify which explicit version of an asset to use for all asset references, or by supplying an additional date-stamp to use when resolving assets.

The FarmAPI is a mechanism that allows users to take charge of the submission of jobs to a render farm and the construction of a look up table might be implemented within this API. See the Farm API docs for how to write a Farm API plug-in.

Setting the Default Asset Management Plug-in

The default Asset plug-in and file sequence is defined with two environment variables. If you want to set your own plug-in and sequence as default, make sure the following are set on your system:

```
KATANA_DEFAULT_ASSET_PLUGIN=yourAssetPlug-in
KATANA_DEFAULT_FILE_SEQUENCE_PLUGIN=yourFileSequencePlug-in
```

The C++ API

You can implement an Asset plug-in in C++ as well as in Python. This is done by inheriting from the FnAsset class in the C++ plug-in SDK. Almost exactly the same methods must be implemented in C++ as in Python.

It is not possible to implement a custom Asset Browser, Asset Control Widget or Asset Render Widget via the C++ plug-in SDK. However, these user interfaces can still be implemented in Python and work alongside a C++ Asset Management Plug-in.

Asset management plug-ins implemented in C++ and Python are accessed via the same Python interface inside of Katana and similarly, C++ plug-ins that make use of an asset management plug-in have access to those implemented in Python and in C++.

In order for Katana to load a custom asset management plug-in, it must be compiled as a shared object and placed in a directory called **Libs** inside your **KATANA_RESOURCES** directory.

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**

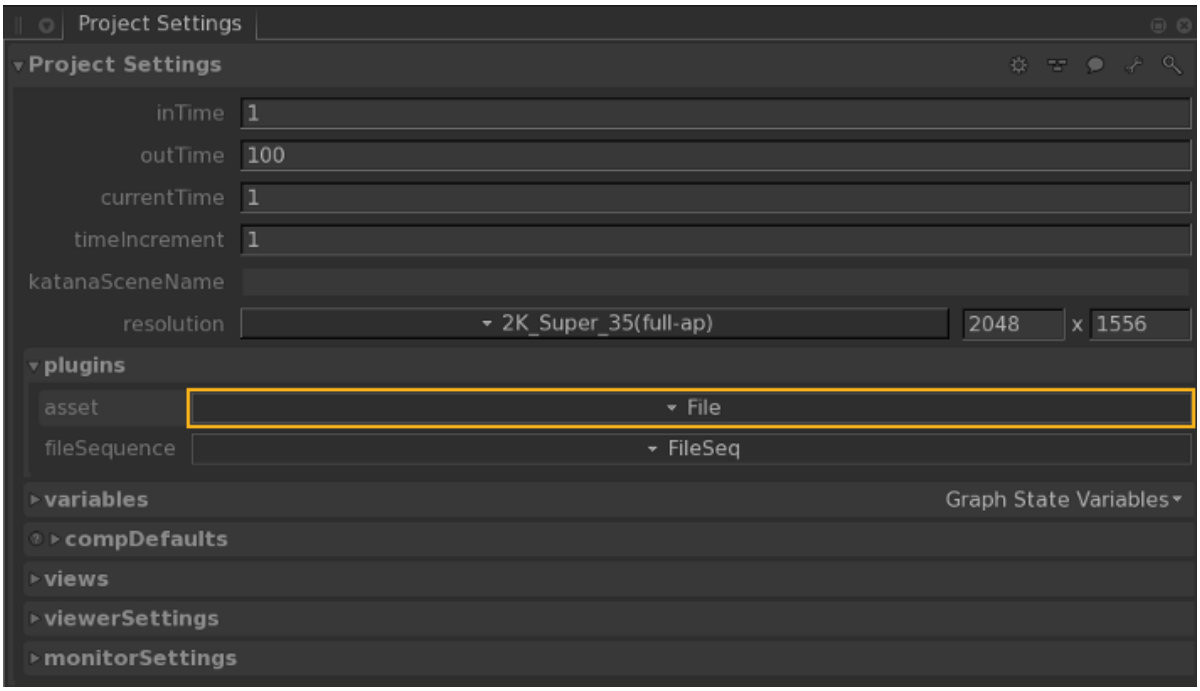
   The asset management system handles storing the asset, and returns the Asset ID actually applied, which can be different to the one supplied in the **Pre-Publish** step. Katana uses that Asset ID from then on to identify the current version of the asset.

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 **plugins** > **asset** and choose a plug-in from the dropdown list.



Note: The default **plugins** > **asset** > **File** option selects Katana's default, manual file management, rather than any asset-managed file management.



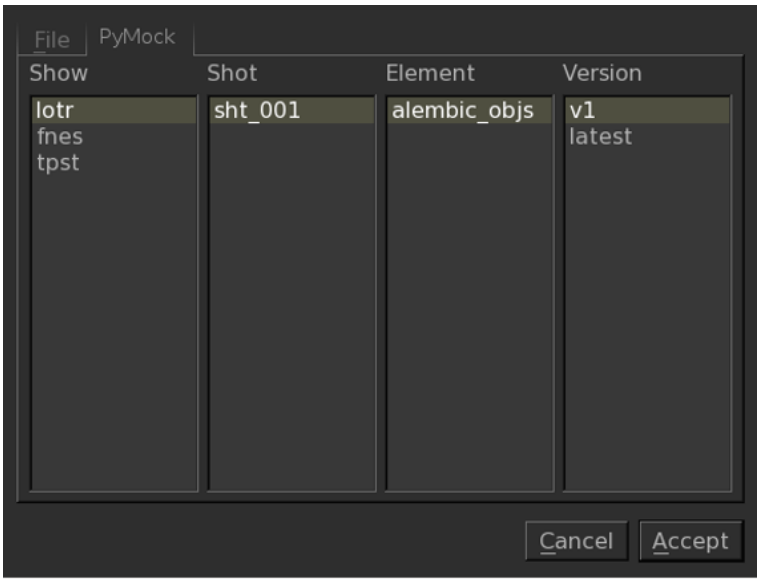
Example Asset Plug-in

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 section show the PyMockAsset plug-in.

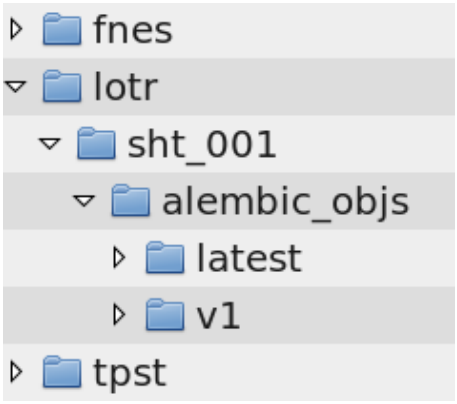
The PyMockAsset plug-in searches for assets under a file location specified in an environment variable called MOCK_ASSET_DIR, for example:

```
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 Python's `tempfile.tempdir` directory
It varies on the platform, see https://docs.python.org/2/library/tempfile.html#tempfile.tempdir.

Retrieve and Publish

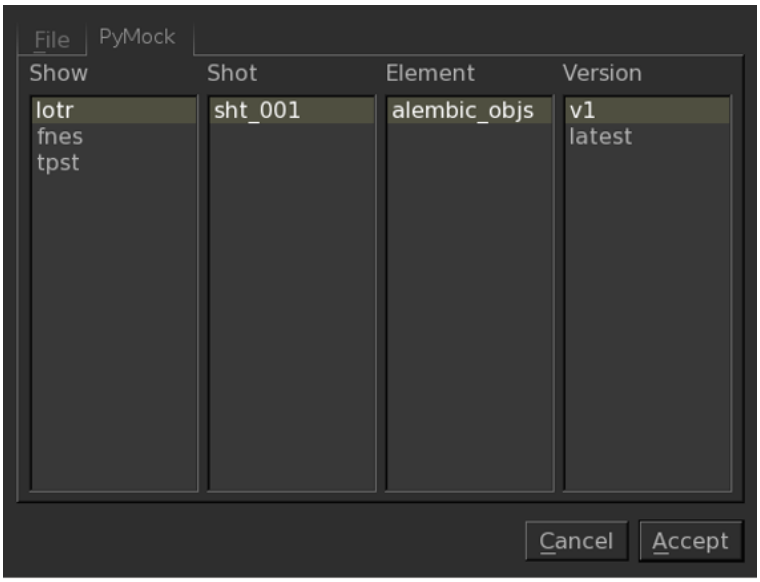
Accessing assets through the UI is performed using the asset browser provided by your plug-in. The browsers used for retrieve and publish can be different. For example, when retrieving assets PyMockAsset shows a browser that only allows selection of existing locations.

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
- 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 Args Files for Shaders in the Dev 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

LiveGroups and LiveShadingGroups

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 Asset Management System Plug-in API.

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 for more information).
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. These unique options can be used to represent the parameter interface of the contents of a LiveGroup node. Nodes inside of the LiveGroup node can reference these user parameters using Python expressions, thus allowing the developer of the LiveGroup's internal node network to make certain parameters of the nodes inside of the LiveGroup available to users of the LiveGroup asset. This effectively exposes the parameters on the outer interface of the LiveGroup node that uses the LiveGroup asset.

When loading a LiveGroup asset into a LiveGroup node, the user parameters of the LiveGroup node are initialized according to the user parameters in the LiveGroup asset. From then on, values and Python expressions on user parameters of the LiveGroup node are stored in the Katana project file itself, or more generally, in the parent node's context.

When reloading a LiveGroup asset from disk into a LiveGroup node, the user parameters are only updated in the sense of adding and removing parameters, but values and Python expressions are not modified as part of a reload. This is done deliberately, so as not to lose any modifications that might have been made to such values and Python expressions on user parameters of LiveGroup nodes.

> **Warning:** Katana doesn't currently determine whether values or Python expressions on user parameters of LiveGroup nodes have been modified intentionally, or left to their default values.
>
> Consequently, changes to values or Python expressions on user parameters of a LiveGroup asset are not propagated to other LiveGroup nodes that use the same LiveGroup asset as their source.

For more information on loading LiveGroup assets into a LiveGroup node, see Loading and Reloading a LiveGroup. Alternatively, for information on accessing or editing any node's parameters, generally, see Editing a Node's Parameters.

In the **Parameters** tab, click the wrench icon, or right-click on the LiveGroup node.

- When the LiveGroup node is in a locked, non-editable state, the following menu options are available:
  - **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, 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.

If a LiveGroup source becomes unavailable, Katana keeps the nodes inside of the LiveGroup node, by default. However, there may be times when you want the contents to be discarded instead. You can achieve this by setting the **KATANA_DISABLE_LIVEGROUP_CACHING** environment variable to 1.

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.

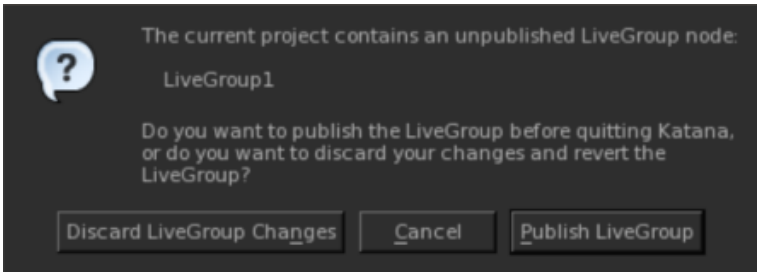
Modified State of Editable LiveGroup Nodes

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.

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.

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.

Share and Reuse Sections of Your Shading Network With LiveShadingGroups

LiveShadingGroups are the material counterpart of LiveGroups and are available in the context of a NetworkMaterialCreate node. LiveShadingGroups allow you to share and version snippets or the entirety of a shading network from one Katana project into another, giving you the ability to create entire material libraries within Katana that can then be shared and reused across projects and departments.



Note: For more information on creating materials and Network Material functionality, see Building Materials Using NetworkMaterialCreate.

LiveShadingGroups are created in the same way and have the same functionality as regular LiveGroups.

Once a LiveShadingGroup is published, the contents can be locked to prevent accidental edits being shared across Katana projects that are using that particular LiveShadingGroup.

In addition, when a LiveShadingGroup is unlocked, changes can be made and published to either overwrite the existing LiveShadingGroup or create a new version. Any changes made to a LiveShadingGroup that is used across different materials or projects are automatically updated.

ShadingGroups can be converted into LiveShadingGroups and shared across Katana projects and between teams. You can also convert LiveShadingGroups into standard ShadingGroups. For more information on ShadingGroups, see Organizing Shading Networks with ShadingGroup Nodes.

For more information on LiveGroups and their functionality, see LiveGroups.

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.

Video: This video demonstrates how Graph State Variables can be used.

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+.** (period). 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.

Scripting and Programming in Katana

Katana utilizes three languages for scripting within the application: Python, Lua, and C++. Each serves a distinct purpose but the language most appropriate for you may vary, depending on what you want to achieve and whether you need to edit the scene graph or scene graph locations.

This page provides an overview of how to get started with scripting and programming in Katana and which language is most appropriate for which tasks.

Python

Python is used widely for rapid application development, especially in the context of APIs to allow users to customize the application. Information about Katana customization can be found in the Katana Developer Guide. For example:

- Working with projects
- Working with nodes
- Customizing node types
- Customizing the user interface
- Python-based parameter expressions

Alternatively, for more information and detailed articles on various Python workflows, visit the Support Portal.

Where fast performance is required, Python isn't always an ideal choice, partly due to the GIL.

In the context of parameter expressions, a faster alternative to Python expressions is Reference Expressions which can be used for simple expressions that reference nodes or parameters. More information on Reference Expressions, can be found in the Katana Developer Guide.

Lua

Lua is used within the OpScript node in Katana. By using OpScript/Lua, it is possible to access the Op API, which is both faster and more powerful than Python. In particular, the OpScript node allows you to modify the structure of the scene graph hierarchy, such as deleting locations, creating new child locations as well as setting and editing attributes.

Lua represents a reasonable balance between fast development turnaround time for developing operations on the scene graph, and fast execution time and stability as part of cooking a scene.



Note: Lua is also useful for prototyping more complex operations that are planned to be implemented as Op Types later.

In certain situations it may be advisable to implement a custom Op type plug-in instead of using OpScript/Lua. Whether this is of advantage should be determined on a case by case basis as it can depend on the complexity of a project and its assets, the number of operations to be performed or the number of

scene graph locations to target. It's advisable to process FX data like particle simulations in Ops/C++, rather than OpScript/Lua.

For an introduction to using OpScript and the Op API, you can look at the OpScript tutorials in available in Katana at:

Help > **Example Projects**

More information on the Op API, see OpScript Nodes or The Op API. You can also refer to Cook Interface (OpScript) in the Katana Developer Guide.

C++

When performance is critical, for example, when working with large data sets, a Lua OpScript can be ported to a C++ Op type plug-in.

Please see the Katana Developer Guide for documentation of the interface. You can also refer to the **HelloWorld** example Op which is shipped with the Katana source code in the following location:
$KATANA_ROOT/plugins/Src/Ops/HelloWorld

Build instructions can be found here:
$KATANA_ROOT/plugins/Src/README.md



Note: For more information and further reading on scripting and programming in Katana, refer to the associated article on the Support Portal, Scripting and Programming in Katana.

Scripting with Python

Python workflow:

1. Enter Python statements in Katana's **Python** tab to perform the required actions.

2. Save your script with the extension **.py** in a directory that is contained in the **sys.path** variable.

3. Later, when you want to execute the same statement sequence, import the **.py** file into Katana's **Python** tab again. Katana executes the statements in the specified order.

For more information on the **Python** tab, see Using the Python Tab, or for detailed information on the Python API click **Help** > **API Reference** > **Python APIs** within the application.

Python on the Web

To read more about Python, check out its documentation, or interact with other Python users. Visit the Python programming language official website at http://www.python.org/.

Shelf Item Scripts

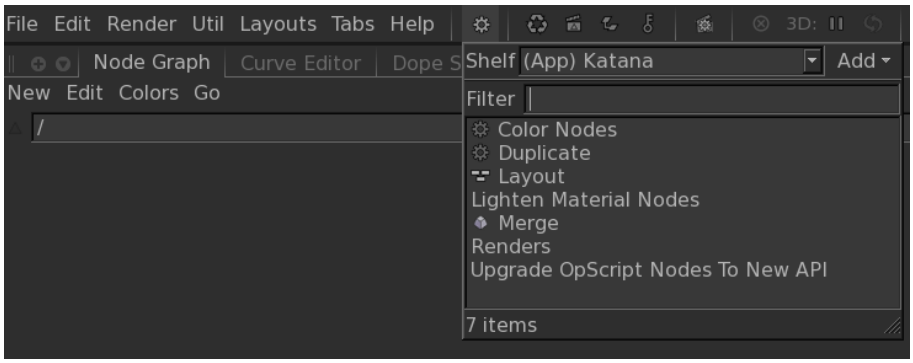
Shelf Item Scripts are Python scripts that you can run from Katana's UI in order to perform arbitrary operations using Katana's APIs. Shelf item scripts implement Shelf Items, which are grouped into Shelves.

Shelf Item Scripts can use Katana's APIs, like the **NodegraphAPI**, to access various parts of Katana that can be queried or modified, for example a project's node graph, with its nodes, parameters, ports, and connections. This section explains how they can be run from Katana's UI, what types of shelves there are, and where the folders and script files that define shelves and shelf items are located.

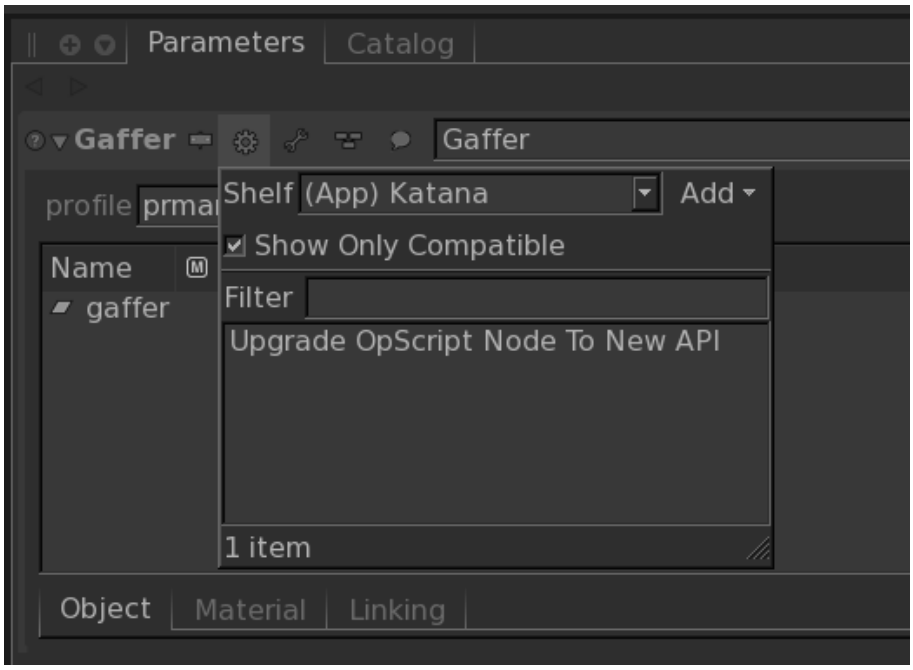
Running Shelf Item Scripts from the UI

The shelf items that are available for you to run from Katana's UI are listed by shelves in the **Shelf Actions** pop-ups that are shown when clicking the **Shelf Actions** toolbar buttons, which appear in different types of toolbars in Katana's UI:

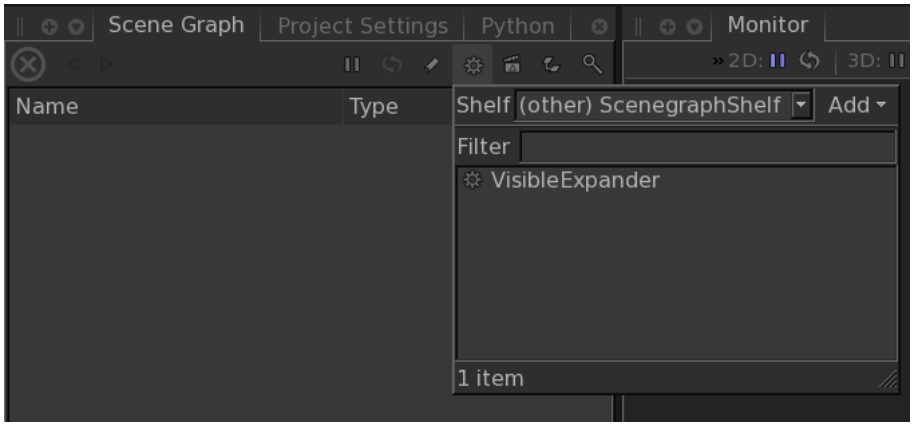
• The toolbar next to the main menu bar in Katana's main window.



• The widgets toolbar for a node that is edited in the **Parameters** tab.



• The toolbar in the **Scene Graph** tab.



The shelves and shelf items listed in the **Shelf Actions** pop-up of the **Parameters** tab are considered to be specific to the type of node that is edited, but this is not enforced: you are free to perform any operation available through Katana APIs or provided by external libraries.

Likewise, the shelves and shelf items listed in the **Shelf Actions** pop-up of the **Scene Graph** tab are considered to be dedicated to working with the scene graph.

Types of Shelves

There are three types of shelves:

• Built-in shelves

• User-defined shelves

• Additional shelves

Built-in Shelves

Built-in shelves contain pre-defined shelf items that ship with Katana releases. In **Shelf Actions** pop-ups, their names are shown with an **(App)** prefix. The shelf item scripts that correspond to shelf items in built-in shelves are loaded from internal Katana resource directories.

You can view the source code of built-in shelf items through the UI, but you can't modify that source code or delete the items or the shelves they are contained in. You can also not create new built-in shelves or shelf items. If you want to create custom shelves and shelf items, use one of the following types of shelves: User-defined Shelves or Additional Shelves.

User-defined Shelves

User-defined shelves are shelves that you can freely create, modify, and delete according to your needs. They are specific to you as a user, so aren't normally available to other artists. They are shown with your username as a prefix, for example **(David)**, if your username is David.

Scripts that implement user-defined shelf items are loaded from the **.katana** directory of your **HOME** folder. You can create and delete shelves for user-defined shelf items, and inside of a shelf, create new shelf items, and edit their source code right from within **Shelf Actions** pop-ups.

Additional Shelves

Additional shelves are shelves that are loaded from directories whose paths are listed in the **KATANA_RESOURCES** environment variable. They can be used to share shelf item scripts between artists across a studio: you can simply place the resource directory with a directory structure as described below in a network location, and add its path to **KATANA_RESOURCES**.

The names of additional shelves are shown with an **(other)** prefix in **Shelf Actions** pop-ups. Just like built-in shelf items, you can view the source code of additional shelf items through the UI, but you can't delete them or modify their source code. You can also not add or remove additional shelves using the UI. They are defined exclusively through script files in shelves folders on disk that are picked up through **KATANA_RESOURCES**.

Directory Structure for Shelf Item Scripts

Within a Katana resource directory, shelves and shelf items are loaded from sub-folders with the following names:

- **Shelves** - contains shelves that are shown in the **Shelf Actions** pop-up in the toolbar of Katana's main window.
- **ShelvesNodeSpecific** - contains shelves that are shown in **Shelf Actions** pop-ups of the **Parameters** tab.
- **ShelvesScenegraph** - contains shelves that are shown in the **Shelf Actions** pop-up of the **Scene Graph** tab.

Sub-folders of those folders represent the Shelves available in **Shelf Actions** pop-ups, and contain the shelf item scripts that correspond to shelf items that are listed for a shelf that is selected in the UI.

Shelf item script files are ASCII text files that use the standard **.py** file extension of Python source files, and can be edited in source code or regular text editor applications.



Note: There is a known issue with loading of shelves where shelf entries of the same name are shown multiple times when multiple shelves of the same name are present in Katana resource directories. The shelves' loading mechanism searches from left to right, and shelves in folders listed later win over shelves in folders that are listed first. If multiple shelves with the same name are present in **KATANA_RESOURCES** directories, all shelf items for that shelf are taken from the last shelf that was loaded with that same name.

Node-Specific Shelf Item Scripts

Node-specific shelf item scripts define shelf items that can be run from the **Shelf Actions** pop-up in the **Parameters** tab for a node whose parameters are shown.

Pre-Defined Variables in Node-Specific Shelf Item Scripts

The following variable is pre-defined for use in node-specific shelf item scripts:

- **node** - The node whose parameters are shown in the **Parameters** tab.



Note: In addition to **node**, other variables may be present, which are set using node interaction delegates.

The following additional variables are defined for certain types of nodes using node interaction delegates:

Nodes	Variables
GafferThree	**selectedItems** - A list of paths of scene graph locations that are selected in the Gaffer table in the parameter interface of those types of nodes.
GroupStack and GroupMerge	**selectedNodes** - A list of nodes that are selected in the parameter interface of those types of nodes.
MaterialStack	**selectedLocations** - A list of paths of scene graph locations that are selected in the parameter interface of MaterialStack nodes.

Targeting Node-Specific Shelf Item Scripts to Specific Types of Nodes

It is possible to add information to node-specific shelf item scripts so that their corresponding shelf items are shown in **Shelf Actions** pop-ups of the **Parameters** tab only for specific types of nodes. A special **SCOPE** field in the docstring of the shelf item script can be used to list names of node types (comma separated) to which the script applies. This information is used in the UI to filter the list of shelf items shown for a selected shelf to only show those shelf items that are compatible with the type of the node whose parameters are shown.

For example, the built-in node-specific shelf item script for updating OpScript nodes contains the following scope information in its docstring:

```
"""
NAME: Upgrade OpScript Node To New API
SCOPE: OpScript

Migrates OpScript nodes from the legacy syntax to the modern syntax.

"""
```

Docstrings of Shelf Item Scripts

User-defined shelf item scripts that are created in the UI use the following module docstring at the top of the file:

```
"""
NAME: <the name of the script to show in the UI>
```

```
ICON: <the filename of icon to use in the UI>
DROP_TYPES: <currently unused>
SCOPE: <names of types of nodes to target by node-specific shelf items>
<description>

"""
```

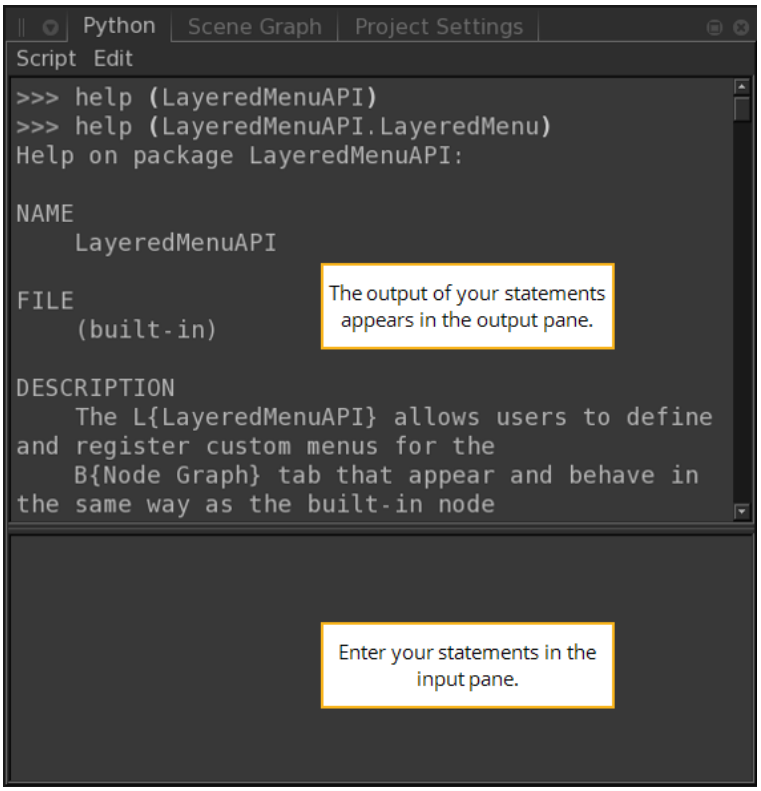
The **SCOPE** field applies to node-specific shelf item scripts only (see section above).

Using the Python Tab

If you're not using a third-party Python interpreter, you can type scripts into Katana's **Python** tab. Open the **Python** tab by clicking the tabs dropdown button or clicking the **Tabs** menu, and selecting **Python** from the list.

Input and Output Panes

The **Python** tab is divided into two parts: the input pane and the output pane. The input pane, the lower part of the tab, is used to type in and execute your Python statements. When you've done this, the statement and the output appears in the output pane, the upper part of the tab.



To adjust the size of either the input or output panes, click and hold the divider between the two and drag it to the desired height. Alternatively, to completely hide either of the panes, click and hold the divider between the two and either drag it all the way to the top or bottom of the tab.

Entering a Statement

To enter a statement in the **Python** tab:

1. Click on the input pane of the editor to insert the cursor there.
2. Type in your statement. To use the usual editing functions, such as copy and paste:
   - Right-click on the editor and select the desired function from the dropdown menu,
   - Click the **Edit** menu in the **Python** tab, and select the desired function from the dropdown menu, or
   - Use the common shortcut for the editing function (for example, to paste a previously-copied statement, press **Ctrl**+**P**).

   When entering the statement, you may notice that any words that are Python's keywords, such as *print* and *import*, turn purple, while strings and comments (content in quotation marks or following a **#**) become green.

```
#This statement prints the words "hello world".

print "hello world"
```

3. If your statement includes several lines, or you want to enter several statements at once, press **Return** to move to the next line.
4. Execute the statement by pressing **Ctrl**+**Return**.



Tip: You can also execute statements by pressing **Ctrl**+**Enter** on the numeric keypad.

By default, successful statements disappear from the input pane, and appear in the output pane. If you enter an invalid statement Katana produces an error in the output pane and clears the statement from the input pane.



Note: Sometimes you may get an error if you copy ad paste statements into the **Python** tab from another source, like an e-mail. 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 manually.

To only execute part of a script, enter the script in the input pane and select the part you want to execute. Press **Ctrl**+**Return**. Katana runs the selected part of the script, leaving the script in the input pane.

If you want to repeat a statement, click **Script** > **History Previous** at the top of the tab, or press **Alt**+**Up**. This moves back to the previous statement. You can do this repeatedly until you reach the statement you want to execute again. Alternatively, if you are further up the script history stack and you want to move back down to a more-recent statement, click **Script** > **History Next** at the top of the tab, or press **Alt**+**Down**. This moves to the next statement down, and you can do this repeatedly until you reach the statement you want to execute. If you are on the last statement in the stack, the input pane is cleared.

To increase the indentation in the input window, press **Tab**. To decrease the indentation in the input window, press **Shift**+**Tab**.

Auto-completion

The **Python** tab provides auto-completion so that you can quickly finish statement tokens. If you write a partial token but are unsure how to complete it or want to fill in the token automatically, press **Tab** while the cursor is still in the input pane.

There are two options for the way in which the auto-completion is handled. You can opt to use the **Shell** or **IDE** auto-completion methods.

- **Shell** - auto-completes the token based on matches with possible completions, when **Tab** is pressed. If there are multiple match possibilities, a list of possibilities is printed to the output pane.
- **IDE** - provides options for possible auto-completion in a pop-up widget as you type. If you want to use a possible match for auto-completion, press the **Up** or **Down** arrows to choose the specific match and then press **Tab** or **Return/Enter** to auto-complete. If you want to close the IDE pop-widget, press **Esc**.

To set the auto-completion method, click **Edit** > **Preferences**, and select the **python** heading in the **Preferences** dialog. Click the **autoCompletionBehavior** dropdown and choose **Shell** or **IDE**.

Clearing the Output Pane

To clear everything that appears in the output pane, click **Edit** > **Clear** at the top of the **Python** tab.



Note: Find out more about the available Python statements in the Katana Developer Guide: Python Expressions.

Automating Procedures

Once you know how to use the **Python** tab to type in a sequence of Python statements, you probably want to learn how to automate the procedure. All you need to do is save your statements, and when you want to use them again later, import them into the **Python** tab.

Importing and Executing a Python Script

To import and execute a Python script in the **Python** tab:

1. On the top of the **Python** tab, click the **Script** > **Source File** menu option.
   The **Run a Script** dialog opens.
2. Navigate to the Python module that contains the script you want to open, or type in the file path to the module, and click **Open**.

OR

In the input pane, enter:

import *module*

Where ***module*** represents the name of your Python module without the file extension, for example:

import firstmodule

Katana imports the Python module and performs the procedure defined in the module.



Note: The statement does not appear in the input pane either before or after it's executed.



Note: Importing the module is done according to Python's default rules. During the import, the module is searched in the following locations and order:

1. In the current directory.
2. In the directories contained in the **PYTHONPATH** environment variable, if this has been defined. To view these directories, enter **echo $PYTHONPATH** in a command shell.
3. In an installation-dependent default directory.

During the search, the variable **sys.path** is initialized from these directories. Modules are then searched in the directories listed by the **sys.path** variable. To see these directories, execute the statement **print sys.path** in the **Python** tab.

Message Logging

Error, warning, and informational messages (to name a few) are logged in Katana using the **logging** module of the **Python Standard Library**. Messages are logged from contexts including the **Python** tab, shelf scripts, and Python startup scripts. Messages shown in the UI are generated by the root logger, which is configured with the **${KATANA_ROOT}/bin/python_log.conf** file.

In addition to the Python logging system, Katana also has a C++ logging system, configured with **${KATANA_ROOT}/bin/log.conf**. In interactive Katana, messages logged using the Python logging module are filtered according to its own configuration and posted in the **Messages** tab, but they are also forwarded to the Log4CPlus logger, configured using the specified (or default) **log.conf**. The Render Log is handled independently.

You can filter the level of messages generated, and the level of messages displayed. For more on how to filter the level of messages generated or displaying messages, see The Message Center.

The C++ **log.conf** logging can be sourced from an external file if this is set through the **FNLOGGING_CONFIG** environment variable.



Note: This new **log.conf** file overrides the internal Katana file, so it needs to be complete to preserve the same print out information. However, using this environment only works for the C++ logging, not for the Python logging driven by the **python_log.conf** file.

Message Levels

Python Message Levels

Katana recognizes the following standard log message levels from the **Python logging** module:

- **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.

Messages shown in the UI are generated by the root logger, which is configured with the **${KATANA_ROOT}/bin/python_log.conf** file. If you're using Katana in Batch mode, all render messages are logged as **INFO** log messages to the **MainBatch** log. The reason for the fixed Python log configuration is that the actual logging is all configurable using **log.conf** instead of **python_log.conf**. The exception to this is the passing of messages to the **Messages** tab, where filtering is performed in an interactive manner.

To configure the Python logger for an interactive session you can acquire the specific logger and set its level. For example, try the following in the **Python** tab:

```
noisyLogger = logging.getLogger('NoisyLogger')
noisyLogger.setLevel(logging.WARN)
```

This should silence all **INFO** messages from this logger only.

For example:

1. Load a scene and note the **INFO** message: [INFO python.NodegraphAPI.NodeXmlIO]: Loading "/tmp/anyScene.katana"...
2. Execute the following in the **Python** tab:

```
loadingLog = logging.getLogger('NodegraphAPI.NodeXmlIO')
loadingLog.setLevel(logging.WARN)
```

3. Now load the scene again.

   The **INFO** message is not logged.

Loggers

There are two ways of logging messages from a Python context:

• directly through the Root Logger, or

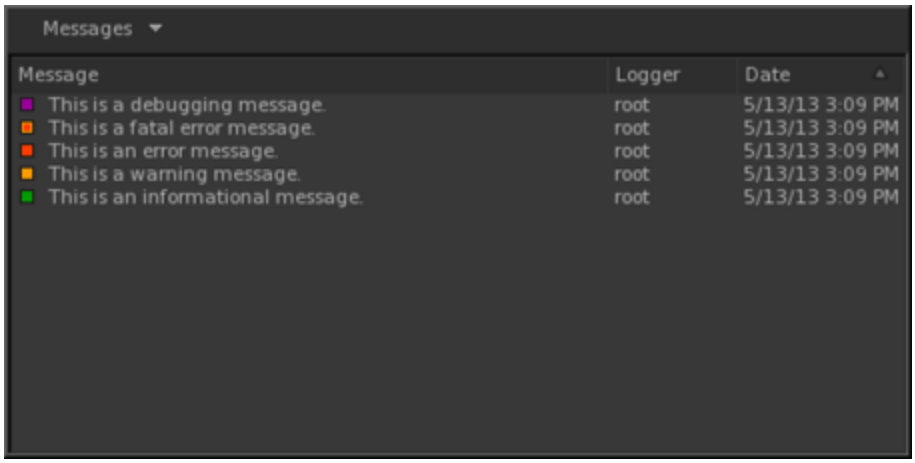
• through a Custom Logger.

Root Logger

The following example logs a message of each supported type through Python's root logger:

```
import logging

logging.info("This is an informational message.")
logging.warning("This is a warning message.")
logging.error("This is an error message.")
logging.critical("This is a fatal error message.")
logging.debug("This is a debugging message.")
```

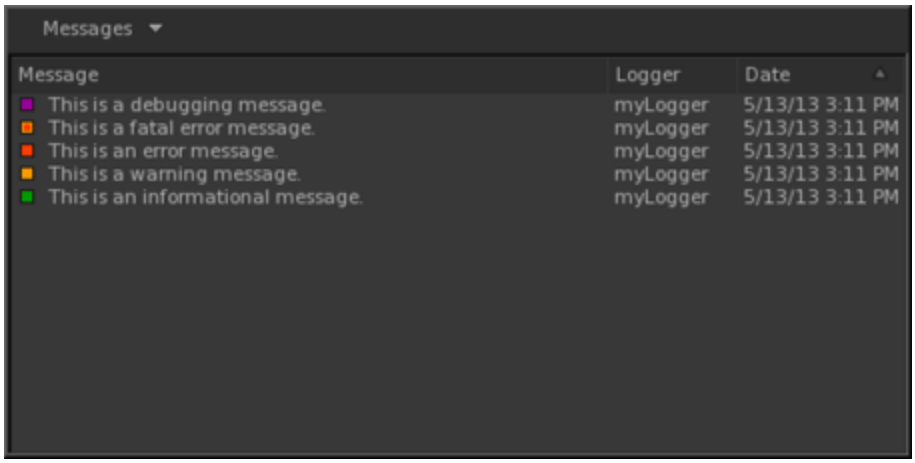


Custom Logger

Instead of using the root logger, you can create a custom logger object. The following example creates a custom logger and generates a message of each level using that logger:

```
import logging

log = logging.getLogger("myLogger")
log.info("This is an informational message.")
log.warning("This is a warning message.")
log.error("This is an error message.")
log.critical("This is a fatal error message.")
log.debug("This is a debugging message.")
```



Note: The message level display option in the **Messages** tab is independent of the level of message actually generated. For example, if the **Messages** tab is set to show debug messages, debug messages are only actually displayed if the level of message generated is also set to include debug. Refer to The Message Center for more information on how to set the level of message that is generated.

```
log4cplus.appender.KatanaConsoleOutput=log4cplus::ConsoleAppender
    log4cplus.appender.KatanaConsoleOutput.layout=log4cplus::PatternLayout
    log4cplus.appender.KatanaConsoleOutput.layout.ConversionPattern=[%p%c]:
%m%n
```

```
log4cplus.appender.KatanaConsoleOutput.layout.ConversionPattern=[%p %c]: %m%n
log4cplus.appender.KatanaConsoleOutput.layout.ConversionPattern=[%p %c %d]:
%m%n
```

Logging Exceptions

Exceptions can be logged in a way that automatically includes traceback information in the log message, as in the following example:

```
import logging

try:
    i = 1 / 0
except Exception as exception:
    logging.exception("Error in computation: %s"
                                          % str(exception))
```

Run in the **Python** tab, this produces a log message with the following text:

```
Error in computation: float division
Traceback (most recent call last):
  File "<string>", line 4, in <module>
ZeroDivisionError: float division
```

Verbose Mode Logging

There is no command-line argument for controlling the verbosity level for console messages. You can modify the type of messages that are appended to the logger by changing the rootLogger details so, for example, to suppress all **INFO** messages you can set **WARN** instead:
`log4cplus.rootLogger=WARN, KatanaLogFile, KatanaConsoleOutput`



Note: To suppress messages in this way, you need to be change the existing **log.conf** file within the Katana installation directory, as there is no direct support for sourcing external **log.conf** files to customize the logging output.

Renderer Logging

Message logging for renderers is specific for each plug-in, and needs to be handled in a **log.conf** file located in the **plugins** directory.



Note: The level set in the **log.conf** core file takes precedence over the level set in your plug-ins. So, setting **log.conf** to **ERROR** and the plug-in log level to **INFO** shows only the errors.

You can also customize the logger completely by creating a new handler for the Logger class, using a similar structure to that below:

```
myLog = logging.getLogger('MyLog')
sh = logging.StreamHandler()
logLevel = logging.WARNING # logging.INFO
sh.setLevel(logLevel)
myLog.addHandler(sh)
```

The Op API

The Op API offers a powerful C++ plug-in interface for manipulating the scene graph and modifying attributes. All of Katana's shipped Ops are written with the Op API. This API allows you to create plug-ins that can arbitrarily create and manipulate scene data. An Op can be given any number of scene graph inputs, inspect the attribute data at any location from those inputs, and can create, delete, and modify the attributes at the current location. Ops can also create and delete child locations, or even delete themselves.

In other words, anything that you can do with any Katana node, you can do with an Op. Examples of the things you can do with Ops include:

- Using context-aware generators and importers,
- Advanced custom merge operations,
- Instancing of hierarchies,
- Building network materials out of fragment parts, and
- Processing to generate geometry for crowds.

Op API Basics

Geolib3 is a library for efficiently loading and processing scene graph data. The Geolib3 scene graph is defined as a hierarchy of named scene graph locations, with each location having a set of named attributes. Scene graph locations are generated and processed on demand to support large data sets.

Example scene graph locations:

```
/root
/root/world/geo/mesh
```

Example attributes:

```
StringAttr("hello world")
FloatAttr([1.0 2.0 3.0 … ])
```

Operators (Ops) are the core processing unit of Geolib3, called upon to compute the scene graph's locations and attributes. Ops can both generate new scene graph locations - the equivalent to Scene Graph Generators and can also process incoming attributes. In fact, Geolib3 Ops are a super-set of both APIs and, in practice, no distinction is made between scene graph generation and modification. The code you need to write for the Op API is also much simpler.

Example Op (Pseudocode):

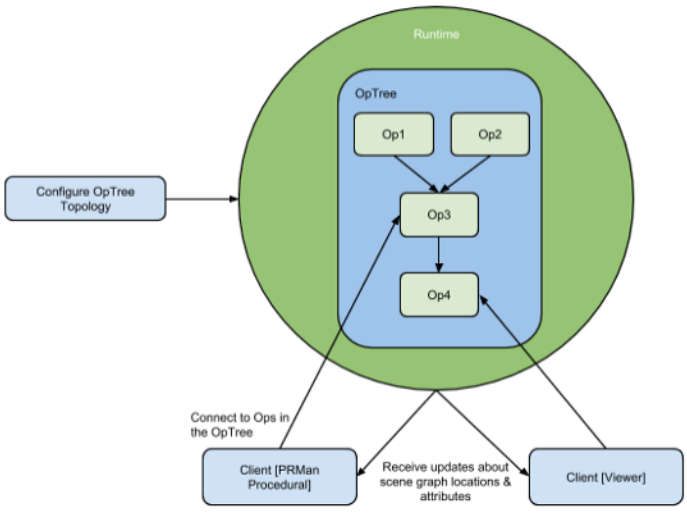
```
attr = getAttribute("taco")
setAttribute("cheese", value)
createChild("world")
```

The OpTree

The tree of connected Operators (the OpTree), is both persistent and mutable. The persistent OpTree allows Katana to inform Geolib3 of only the changes to the OpTree's topology and arguments, rather than having to describe the complete set of Ops again from scratch. This persistent OpTree is efficient by not only allowing simpler update mechanisms when only a sub-set of Ops have changed, but is also more efficient from a computational standpoint, as the underlying engine can potentially reuse previously computed (and cached) results.

Katana cannot directly query from arbitrary Ops in the OpTree. Instead, Clients are created and pointed at an Op. Locations and attributes, which represent the cumulative result of the upstream OpTree, can then be computed upon request.

The Runtime is the underlying computational engine responsible for maintaining the persistent representation of the OpTree, scheduling Op execution, and delivering results to Clients. The runtime can be used either in a synchronous or asynchronous manner. The synchronous interaction model is common at render time while the asynchronous model is common during UI interaction.



Core Concepts with Geolib3

There are three core concepts in Geolib3 that pertains to the Op API: the Runtime, Ops, and Clients. In the sections below, we'll address the host of the system - the Runtime - and the services it provides before looking closely at Ops and the concept of the Client, and its use.

Geolib3: Into the Details

Geolib3 is Katana's new deferred scene graph processing library. Geolib3 works at Katana's core, processing and generating scene graph locations on demand, to support large data sets. Geolib3 supports an asynchronous processing model allowing the UI to remain responsive while scene graph data is being processed.

Operators (Ops) are the core processing unit of Geolib3. Ops can both generate new scene graph locations (equivalent to Geolib2 Scene Graph Generators) and process incoming attributes.

Katana uses Clients to query attributes on specific locations when requested by the UI, for example to show attribute values in the **Attributes** tab, or during rendering, when the scene graph is traversed and processed to deliver data to the selected renderer.

Differences Between Geolib2 and Geolib3

- Geolib2 did not have a persistent scene graph data model. Conceptually, the entire scene graph is reconstructed on every edit. Conversely, Geolib3's OpTree is persistent, allowing for inter-cook scene data re-use.
- Geolib2's scene graph was traversed using an implicit index mechanism, for example **getFirstChild()**, **getNextSibling()**, with scene graph location names determined by the **name** attribute. In Geolib3, children are natively indexed by name. Thus, in Geolib3 you can selectively cook a location, by name, without cooking any peers. Consequently, the **name** attribute is meaningless. However, this also implies that locations cannot rename themselves (you can rename children, however).
- Geolib2 was not amenable to either asynchronous or concurrent evaluation. Geolib3 supports both of these features.

The Runtime

The Runtime is responsible for coordinating Op execution, and provides a few key services:

- A means of configuring and modifying the persistent OpTree. This includes creating instances of Ops, connecting Ops to each other, and providing them with arguments to determine their behavior. Within

Katana, artists interact with nodes rather than the OpTree directly. There is roughly a 1:1 correspondence between nodes in the node graph and Ops in the OpTree.

- The ability to register your interest in specific scene graph locations and their attributes that are produced as a result of evaluating the OpTree.

Internally, the Runtime has a number of other responsibilities including:

- Managing the scheduling and evaluation of Ops.
- Observing dependencies between Ops to ensure correct scene graph generation.
- Caching of location and attribute data for retrieval.
- Distribution of location and attribute data to clients.

The Runtime is able to use all the information it gathers from your interactions with it to efficiently manage resources. For example, if you don't attach any Clients to the OpTree then it does not need to evaluate any Ops or, if no dependencies exist between two Ops, it can concurrently schedule their evaluation to make best use of multicore systems.

From a technical perspective, you can interact with the Runtime through a C++ or Python interface, which provides a great deal of flexibility in how you configure your OpTree and listen to scene graph updates.

Interface	Languages Available
Ops	C++ and Lua (with OpScript)
Client Configuration	C++ and Python
OpTree Configuration	C++ and Python

Ops

A Katana Geolib3 Op is the lowest level scene graph processing “unit”, responsible for building or processing scene graph data on demand. All scene graph loading/processing functionality internal to Katana is implemented using the same Op API available for your own custom development. Conceptually, Ops are a super-set of the old geometry APIs in Katana, including Scene Graph Generators.

Examples of what Ops can do include:

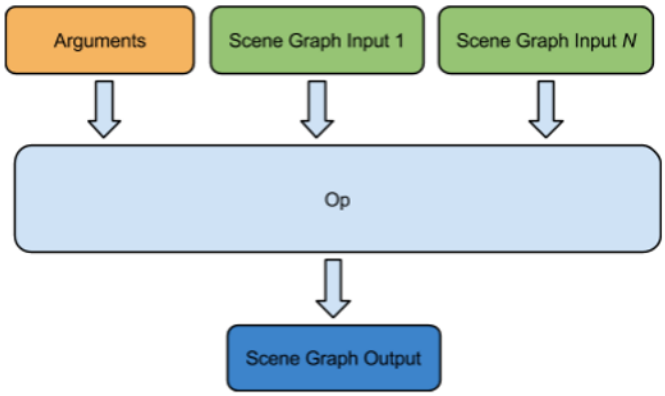
- Setting attributes
- Creating child scene graph locations
- Deleting child scene graph locations
- Getting attributes from the incoming scene graph
- Getting the available Op arguments.

Rules for Ops include:

- The OpTree defines the connectivity for what an Op sees as its input.
- Each Op is responsible for registering a stateless function, which is called on demand at all locations on the input scene graph. This function is also referred to, later on, as the Cook function.
- For Ops that do not have an input, the function is called at the root location giving them the opportunity to construct a more complex scene graph.
- From the perspective of an Op implementer, the incoming scene is immutable. Only the output scene can be modified.
- Although an Op can run on many different locations, it's called separately for each location. Each time an Op is called, the result is a single scene graph location - the 'output location'.
- Ops are expected in their implementation to do the minimum amount of work necessary to produce the specified scene graph location, in order to be a "good citizen" in a deferred processing system.
- Roughly speaking, when a downstream client is evaluated, all upstream Ops in the OpTree are run over all scene graph locations that exist (and are expanded) in the incoming tree. While there are more sophisticated API calls to change which Op runs at child locations, substituting out your OpArgs, the Optype, or even calling into another Op entirely, these can be ignored during your initial exposure to Op writing.
- An Op is evaluated from a starting location. This is usually the familiar **/root**, however, Geolib3 provides mechanisms that allow you to redefine an Op's starting location. The ability to change an Op's starting location is extremely powerful and allows you to write Ops than can work either relative to your start location or in a more absolute manner.

Ops have two types of input:

1. Op arguments - these are provided by the user to govern how the Op behaves when evaluated at a particular scene graph location. When you instantiate an Op you provide a set of root location arguments, which are the arguments the Op receives when run at its starting location. For instance, parameter values from nodes and system args, such as the current frame time, are passed to Ops using Op Arguments.
2. Scene graph input(s) - locations and attributes that have been produced by other upstream Ops in the OpTree, which are connected to the Op currently being evaluated, are available as input and query-able in a read-only state.



The Runtime evaluates your Op at, potentially many, scene graph locations and it is up to you, the Op Writer, to determine the action taken at any particular location. As an Op Writer, you have access to a rich API, whose functionality can be broken down into three areas:

- Functions to interrogate the scene graph location the Op is currently being evaluated at.
- Functions to interrogate the state of the connected incoming scene graph.
- Functions to modify the output scene graph as a result of evaluating the Op at a given location. In addition to changing the output scene graph, it is also possible for an Op to change, at evaluation time, what Op and corresponding arguments are evaluated at child locations. It's also possible for an Op to arbitrarily execute other Ops during its evaluation.

Clients

In order to view the scene graph locations and attributes generated as a result of evaluating Ops, such as to walk the scene graph to declare data to a renderer, or to inspect the values in the **Attributes** tab, we use Clients. A Client is connected to a specific Op in the OpTree and, in this context, we refer to it as a **Terminal Op**. We can control the scene graph locations we are interested in receiving updates for using the Client API.

To ensure the Runtime re-computes scene graph locations for every commit, set these locations as **active**. To ensure the Runtime re-computes a scene graph location's children whenever it is cooked, set the location as **open**. As an extension to the **open** state, you can set a location to **recursive open**, which also sets the child locations produced as a result of evaluating that location to **open** in a recursive manner. This provides behavior the same as the existing **forceExpand** option. To conduct a one-shot computation of a scene graph location, you can instruct the Runtime to **ready** it.

Once you have created a Client, connected to a specific Op, you can then declare locations in the scene graph that you are interested in getting data from. The client then receives events from the Geolib3 Runtime when the requested data is ready.

Examples of Clients include:

- The **Attributes** tab - sets a single location in the **Scene Graph** tab as **active**. When anything at that location is changed the **Attributes** tab is notified of the updates.
- The **Scene Graph** tab - sets **/root** as active. As you open locations in the UI, these locations are set to **open** on the Client. On subsequent updates to the OpTree, open portions of the scene graph are automatically recomputed.
- Renderers - typically make repeated calls to the Client to **ready** a location, read the required attributes, declare them to the renderer's API, then immediately discard the data.

Client Configuration

You can point a Client at a particular Op, configure it to listen to particular locations, and interrogate the scene graph locations and attributes that are returned by the Runtime. With client configuration, you are also able to use transactions, which are objects used to batch together operations that are submitted to the Runtime at one time.

The first step in Client configuration is the setup of the client with the Runtime. To do this:

```
# Create the Runtime and a transaction to batch our work for the Runtime
into
runtime = FnGeolib.GetRegisteredRuntimeInstance()
transaction = runtime.createTransaction()

# Create the Client and point it a terminalOp
client = transaction.createClient()
transaction.setClientOp(client, terminalOp)

# Push these changes to the Runtime
runtime.commit([transaction,])

# Set a location we're interested in as active
client.setLocationsActive(('/root/world',))
```

We can then ask the Client for information about the locations we've registered interest in, at any point:

```
# Get the list of changed locations
locationDataChangeEvents = client.getLocationEvents()
if not locationDataChangeEvents:
    return

# Iterate over each location we've been informed about and interrogate it
for event in locationDataChangeEvents:
    location = event.getLocationPath()
```

```
    locationData = event.getLocationData()
    locationAttrs = locationData.getAttrs()
    if not isinstance(locationAttrs, FnAttribute.GroupAttribute):
        continue

# Only look at locations of type 'light'
typeAttr = locationAttrs.getChildByName('type')
if (not isinstance(typeAttr, FnAttribute.StringAttribute)
    or typeAttr.getValue('', False) != 'light'):
    continue

# Only want to look at xform or material updates
for attrName in ('xform.matrix', 'material',):
    attr = locationAttrs.getChildByName(attrName)

# Do something with attr
```

The OP API Explained

This section covers the following elements of the Op API:

- **The cook interface**, what it is, and how it fits into Geolib3.
- **Op arguments** and modifying arguments that are passed down to children.
- **Scene graph creation** and hierarchy topology management, including how to create and delete scene graph locations, and controlling where an Op is executed.
- **Reading scene graph input** from potentially many inputs, and the associated issues.
- **CEL** and other utility functions that are available to you, as an Op writer, to accomplish common tasks.
- **Integrating your Op** with the node graph.

You can find concrete examples of the above concepts in the **$KATANA_HOME/plugins/Src/Ops** directory where the source code for a number of core Ops is kept. Below is a brief overview of some of these Ops, and examples of where they are currently used:

- AttributeCopy - provides the implementation for the AttributeCopy node, which copies attributes at locations from one branch of a scene to another.
- AttributeSet - the back-end to the AttributeSet node, it allows you to set, change, and delete attributes at arbitrary locations in the incoming scene graph.
- HierarchyCopy - like the AttributeSet Op, it's the back-end to the HierarchyCopy node, allowing you to copy arbitrary portions of scene graph hierarchy to other parts of the scene graph.
- Prune - removes any locations that match the CEL expression you provide from the scene.

- StaticSceneCreate - produces a static hierarchy based on a set of arguments you provide. This Op is the core of HierarchyCreate, and is used extensively by other Ops and nodes to produce the hierarchies of locations and attributes that they need. For example, the CameraCreate node uses a StaticSceneCreate Op to produce the required hierarchy for a camera location.

The Cook Interface

The cook interface is the interface Geolib3 provides to implement your Op's functionality. You are passed a valid instance of this interface when your Op's **cook()** method is called. As discussed above, this interface provides methods that allow you to interrogate arguments, create or modify scene graph topology, and read scene graph input. You can find a full list of the available methods on the cook interface in **$KATANA_HOME/plugin_apis/include/FnGeolib/op/FnGeolibCookInterface.h.**



Note: See more in the Katana Developer Guide: Cook Interface (OpScript)

Op Arguments

As discussed previously, Ops are provided with two forms of input: scene graph input created by upstream Ops and Op arguments, which are passed to the Op to configure how it should run. Examples of user arguments include CEL statements describing the locations where the Op should run, a file path pointing to a geometry cache that should be loaded, or a list of child locations the Op should create.

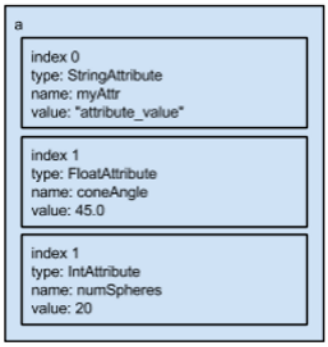
We'll first look at the simple case of interrogating arguments and then look at a common pattern of recursively passing arguments down to child locations.

Reading Arguments

Arguments are passed to your Op as instances of the **FnAttribute** class. The cook interface has the following function call to retrieve Op arguments:

```
FnAttribute::Attribute getOpArg(
    const std::string& specificArgName = std::string()) const;
```

For example, the StaticSceneCreate Op accepts a GroupAttribute called **a** that contains a list of attributes, which contain values to set at a given location. This appears as:



StaticSceneCreate handles the **a** argument as follows:

```
FnAttribute::GroupAttribute a = interface.getOpArg("a");
if (a.isValid())
{
    for (int i = 0; i < a.getNumberOfChildren(); ++i)
    {
        interface.setAttr(a.getChildName(i), a.getChildByIndex(i));
    }
}
```

Note: It's important to check the validity of an attribute after retrieving it using the **isValid()** call. You should check an attribute's validity every time you are returned an attribute from the cook interface.

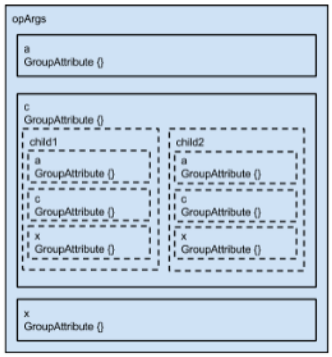
Passing Arguments to Child Locations

There is a common recursive approach to passing arguments down to child locations on which an Op runs. The StaticSceneCreate Op exemplifies this pattern quite nicely.

StaticSceneCreate sets attributes and creates a hierarchy of child locations based on the value of one of the arguments passed to it. This argument is a GroupAttribute that for each location describes:

- **a** - attributes values
- **c** - the names of any child locations
- **x** - whether an additional Op needs to be evaluated at that location

To pass arguments to the children it creates, it peels off the lower layers of the **c** argument and passes them to its children. Conceptually, you can consider it as follows (details of **a** and **x** are omitted for brevity):



The Op running at its current location reads **a**, **c**, and **x**. For each child GroupAttribute of **c** it creates a new child location with the GroupAttribute's name, for example, child1/child2, and pass the GroupAttribute as that location's arguments.

Creating a child in code makes use of the following key function call:

```
void createChild(const std::string& name,
                 const std::string& optype = "",
                 const FnAttribute::Attribute& args = FnAttribute::Attribute(),
                 ResetRoot resetRoot = ResetRootAuto,
                 void* privateData = 0x0,
                 void (*deletePrivateData)(void* data) = 0x0);
```

The **createChild()** function creates a child of the location where the Op is being evaluated at. The function also instructs the Runtime the type of Op that should run there (by default, the same type of Op as the Op that called **createChild()**) and the arguments that should be passed to it. In StaticSceneCreate this looks as follows:

```
for (int childindex = 0; childindex < c.getNumberOfChildren(); ++childindex)
{
    std::string childName = c.getChildName(childindex);
    FnAttribute::GroupAttribute childArgs = c.getChildByIndex(childindex);
    interface.createChild(childName, "", childArgs);
}
```

Scene Graph Creation

One of the main tasks of an Op is to produce scene graph locations and attributes. The Op API offers a rich set of functionality in order to do this. There are five key functions that can be used to modify scene graph topology and control Op execution, which we'll explain below.



Note: It is important to remember the distinction between the set of functions described here and those described in Reading Scene Graph Input. All functions described here operate on the **output of an Op at a given Scene Graph location**. The functions described in Reading Scene Graph Input relate only to reading the scene graph data on the **input** of an Op at a given scene graph location, which is immutable.

The setAttr() Function

```
void setAttr(const std::string& attrName,
             const FnAttribute::Attribute& value,
             const bool groupInherit = true);
```

The **setAttr()** function allows you to set an attribute value at the location at which your Op is currently being evaluated. For example, to set a **StringAttribute** at your Op's root location you can do the following:

```
if (interface.atRoot())
{
    interface.setAttr("myAttr", FnAttribute::StringAttribute("Val"));
}
```

It is not possible to set attributes at locations other than those where your Op is currently being evaluated. If you call **setAttr()** for a given attribute name multiple times on the same location, the last one called is the one that is used. The **groupInherit** parameter is used to determine if the attribute should be inherited by its children.



Note: Since **setAttr()** sets values on the Op's output, while **getAttr()** is reading immutable values on a given input, if a call to **setAttr()** is followed immediately by **getAttr()**, the result is still just the value from the relevant input, rather than returning the value set by the **setAttr()**.

The createChild() Function

```
void createChild(const std::string& name,
                 const std::string& optype = "",
                 const FnAttribute::Attribute& args = FnAttribute::Attribute(),
                 ResetRoot resetRoot = ResetRootAuto,
                 void* privateData = 0x0,
                 void (*deletePrivateData)(void* data) = 0x0);
```

The **createChild()** function allows you to create children under the scene graph location at which your Op is being evaluated. In the simplest case it requires the name of the location to create, and arguments that should be passed to the Op that is evaluated at that location. For example:

```
interface.createChild(childName, "", childArgs);
```

If you specify optype as an empty string, the same Op that called create child is evaluated at the child location. However, you can specify any other optype and that is run instead.



Note: Multiple calls to **createChild()** for the same named child location causes the last specified optype to be used, that is to say, successive calls to **createChild()** mask prior calls.

The **resetRoot** parameter takes one of three values:

- **ResetRootTrue** - the root location of the Op evaluated at the new location is reset to the new location path.
- **ResetRootAuto** (the default) - the root location is reset only if optype is different to the Op calling **createChild()**.
- **ResetRootFalse** - the root location of the Op evaluated at the new location is inherited from the Op that called **createChild()**.

This parameter controls what is used as the **rootLocation** for the Op when it is run at the child location.

The execOp() Function

```
void execOp(const std::string& opType,
            const FnAttribute::GroupAttribute& args);
```

By the time the Geolib3 Runtime comes to evaluating the OpTree, it is static and fixed. The cook interface provides a number of functions, which allow you to request that Ops that were not declared when the OpTree was constructed, be executed during evaluation time of the OpTree.

We have already seen how **createChild()** allows you to do this by allowing you to specify which Op is run at the child location. The **execOp()** function allows an Op to directly call the execution of another Op, providing another mechanism to evaluate Ops, which are not directly declared in the original OpTree. This differs from the **createChild()** behavior, where we declare a different Op to run at child locations in a number of ways, including that:

- It should be thought of as a one-shot execution of another Op, and
- The Op specified in the **execOp()** call is evaluated as if it were being run at the same location with the same root location as the caller.

You can see **execOp()** in action in the StaticSceneCreate Op, where Op types are specified in the **x** argument:

```
// Exec some ops?
FnAttribute::GroupAttribute opGroups = interface.getOpArg("x");
if (opGroups.isValid())
{
    for (int childindex = 0; childindex < opGroups.getNumberOfChildren();
         ++childindex)
    {
         ...
         if (!opType.isValid() || !opArgs.isValid())
         {
             continue;
         }
         interface.execOp(opType.getValue("", false), opArgs);
     }
}
```

The deleteSelf() Function

```
void deleteSelf();
```

Thus far, we have only seen mechanisms to add data to the scene graph, but the **deleteSelf()** function and the associated function **deleteChild()** allow you to remove locations from the scene graph. Their behavior is self-explanatory but their side effects are less intuitive and are explained fully in Reading Scene Graph Input. For now, however, an example for what a Prune Op may look like by using the **deleteSelf()** function call is shown below:

```
// Use CEL Utility function to evaluate CEL expression
FnAttribute::StringAttribute celAttr = interface.getOpArg("CEL");
if (!celAttr.isValid())
    return;

Foundry::Katana::MatchesCELInfo info;
Foundry::Katana::MatchesCEL(info, interface, celAttr);

if (!info.matches)
    return;
// Otherwise, delete myself
interface.deleteSelf();

return;
```

The stopChildTraversal() Function

```
void stopChildTraversal();
```

The **stopChildTraversal()** function is one of the functions that allows you to control on which locations your Op is run. It stops the execution of this Op at any child of the current location being evaluated. It is best explained by way of example.

Say we have an input scene:

```
/root
    /world
        /light
```

Say what we want is:

```
/root
    /world
        /geo
            /taco
        /light
```

So we use a StaticSceneCreate Op to create this additional hierarchy at the starting location /root/world:

```
/geo
    /taco
```

However, if we don't call **stopChildTraversal()** when the StaticSceneCreate Op is at **/root/world** then this Op is run at both **/root/world** and **/root/world/light**, resulting in:

```
/root
    /world
        /geo
            /taco
        /light
            /geo
                /taco
```

To summarize, **stopChildTraversal()** stops your Op from being automatically evaluated at any of the child locations that exist on its input. The most common use of **stopChildTraversal()** is for efficiency. If we can determine, for example, by looking at a CEL expression, that this Op has no effect at any locations deeper in the hierarchy than the current one, it's good practice to call **stopChildTraversal()** so that we don't even call this Op on any child locations.

Reading Scene Graph Input

There are a range of functions that read the input scene graph produced by upstream Ops. All these functions allow only read functionality; the input scene is immutable.

The getNumInputs() function

```
int getNumInputs() const;
```

An Op can have the output from multiple other Ops as its input. Obvious use cases for this are instances where you wish to merge multiple scene graphs produced by different OpTrees into a single scene graph, comparing attribute values in two scene graph states, or copying one scene graph into another one. The **getNumInputs()** function allows you to determine how many Ops you have as inputs, which is a precursor to interrogating different branches of the OpTree for scene graph data.



Warning: It is worth noting that, given the deferred processing model of Geolib3, the "get" functions, such as **getAttr()**, **getPotentialChildren()**, **doesInputExist()**, may ask for scene graph information that has not yet been computed.

In such this instance, your Op's execution is aborted (using an exception) and re-scheduled when the requested location is ready. Thus, Op writers should not attempt to blindly catch all exceptions with "(...)" and, furthermore, should attempt to write exception-safe code.

If a user Op does accidentally catch one of these exceptions, the runtime detects this and considers the results invalid, generating an error in the scene graph.

If your Op is only reading from its default input location (and index) or its parents, "recooks" are unlikely to occur. However, for scattered access queries, either on the input location path or on the input index, "recooks" are likely. If an Op needs to do scattered access queries from a multitude of locations, which would otherwise have unfortunate performance characteristics, an API call - **prefetch()** - is available and is discussed in further detail later on.

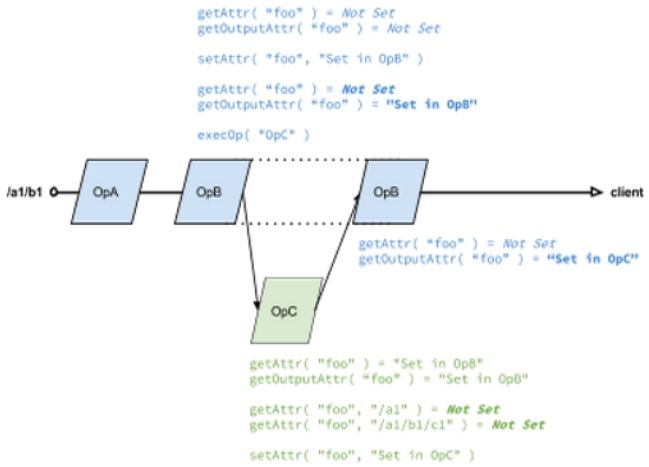
The getAttr() Function

```
FnAttribute::Attribute getAttr(
    const std::string& attrName,
    const std::string& inputLocationPath = std::string(),
    int inputIndex = kFnKatGeolibDefaultInput) const;
```

It is often necessary to perform some action or compute a value based on the result stored in another attribute. The **getAttr()** function allows you to interrogate any part of the incoming scene graph by providing the attribute name and a scene graph location path (either absolute or relative). Additionally, you can specify a particular input index to obtain the attribute value from, which must be smaller than the result of **getNumInputs()**. It is important to note that getAttr always returns the value as seen at the input to the

Op. If you wish to consider any setAttrs already made, either by yourself or another Op invoked with execOp, you must use getOutputAttr.

The following diagram illustrates some of the subtleties of this and, most importantly, that getAttr in an execOp Op, only sees the results of the calling Op when the query location is the current location, otherwise you see the input to the calling Op's 'slot' in the Op graph.



The getPotentialChildren() Function

```
FnAttribute::StringAttribute getPotentialChildren(
    const std::string& inputLocationPath = std::string(),
    int inputIndex = kFnKatGeolibDefaultInput) const;
```

In Scene Graph Creation the function **deleteSelf()** was introduced, noting that the consequence of such a call is more subtle than it may have first appeared. When an upstream Op is evaluated and creates children, if downstream Ops have the ability to delete them, the upstream Op can only go so far as to state that the children it creates may potentially exist after a downstream Op has been evaluated at those child locations. This is because the Op has no knowledge of what a downstream Op may do when evaluated at such a location. To that extent, **getPotentialChildren()** returns a list of all the children of a given location on the input of an Op.

The prefetch() Function

```
void prefetch(const std::string& inputLocationPath = std::string(),
              int inputIndex = kFnKatGeolibDefaultInput) const;
```

Maintain good code practice by using prefetch() as early as possible in the code for your Op's cook function.

The prefetch() function can be used to indicate dependencies between scene graph locations. For example:

/root/world/geo/a may call prefetch() for **/root/world/geo/b** because during the processing of **/root/world/geo/a**, attributes present at **./b** will be required.

CEL and Utilities

There are a number of tasks that Ops are frequently required to complete, such as:

- Creating a hierarchy of locations,
- Determining whether an Op should run based on a CEL expression argument,
- Reporting errors to the user through the scene graph, and
- Obtaining well-known attribute values in an easy to use format, for example, bounding boxes.

Currently you can find headers for these utilities in:

- **$KATANA_HOME/plugin_apis/include/FnGeolib/op/FnGeolibCookInterface.h**
- **$KATANA_HOME/plugin_apis/include/FnGeolib/util/***

The utility implementations live in:

- **$KATANA_HOME/plugin_apis/src/FnGeolib/op/FnGeolibCookInterfaceUtils.cpp**
- **$KATANA_HOME/plugin_apis/src/FnGeolib/util/***

Many of these utilities are self-documenting and follow similar patterns. The following example demonstrates using the CEL matching utilities:

```
// Should we run here? If not, return.
FnAttribute::StringAttribute celAttr = interface.getOpArg("CEL");
if (!celAttr.isValid())
    return;
Foundry::Katana::MatchesCELInfo info;
Foundry::Katana::MatchesCEL(info, interface, celAttr);
if (!info.canMatchChildren)
{
    interface.stopChildTraversal();
}
if (!info.matches)
    return;
```

In the example above, a couple of things are achieved:

1. We determine whether the CEL expression could potentially match any of the children, and if not, we direct the Runtime to not evaluate this Op at child locations.
2. We determine whether we should run at this location, and return early if not.

Feel free to explore the range of utility functions available, as it can increase your productivity when writing Ops.

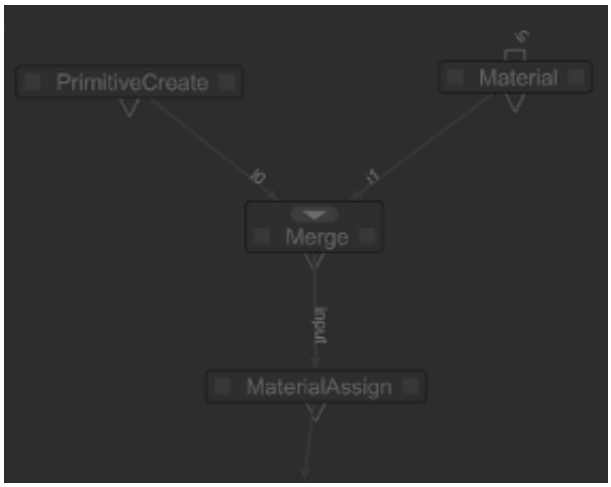
Groups, Macros, and SuperTools

Groups, Macros, and SuperTools allow you to create higher-level compound nodes out of other nodes. Groups and macros are created in the Katana UI, while SuperTools are created using Python.

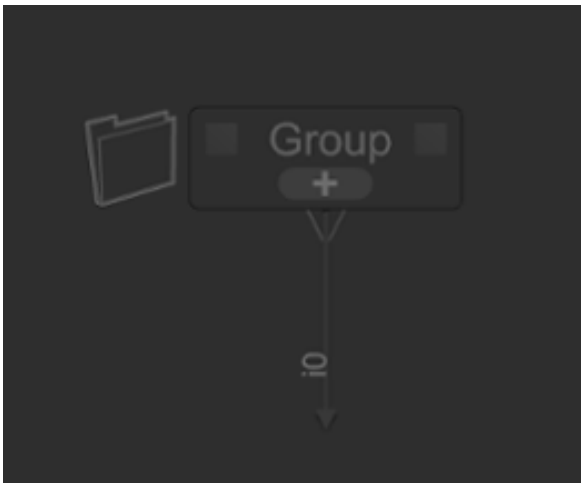
Introduction to Groups

Groups are simply nodes that group together a number of other nodes into a single node. 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 **G**. A new Group node is created, with the previously selected nodes as its children. Any connections between selected nodes are preserved. You can also create an empty group by choosing **Group** from the **Tab** node creation menu. Group nodes can also be duplicated like any other node, creating duplicates of any child nodes and preserving any connections between them.

A Group node may have upstream and downstream connections, depending on the connections of the nodes contained within it.

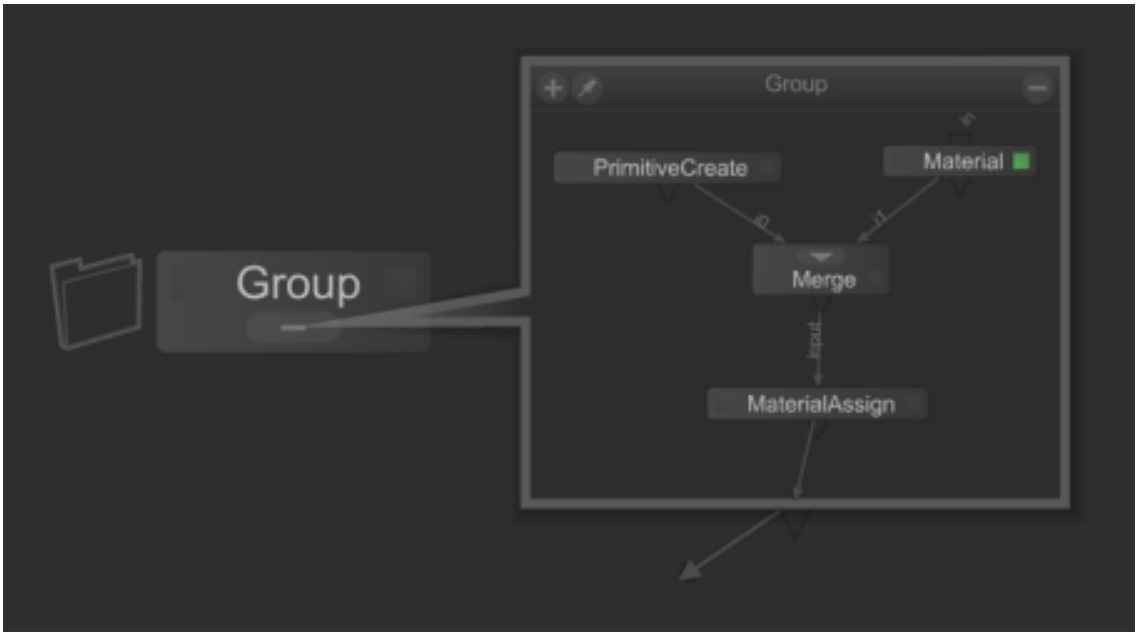


The contents of a group that holds PrimitiveCreate, Material, Merge, and MaterialAssign nodes, with a connection leaving the group from the MaterialAssign node.



None of the nodes in the group have incoming connections from outside the group. Nodes in the group with outgoing connections show an output connection from the Group node.

To see and edit the nodes within a group, click on the + icon on the node to show the group's contents in a new window. When the new window is open, the + icon on the Group node turns into a − sign. Clicking the − sign closes the new window.



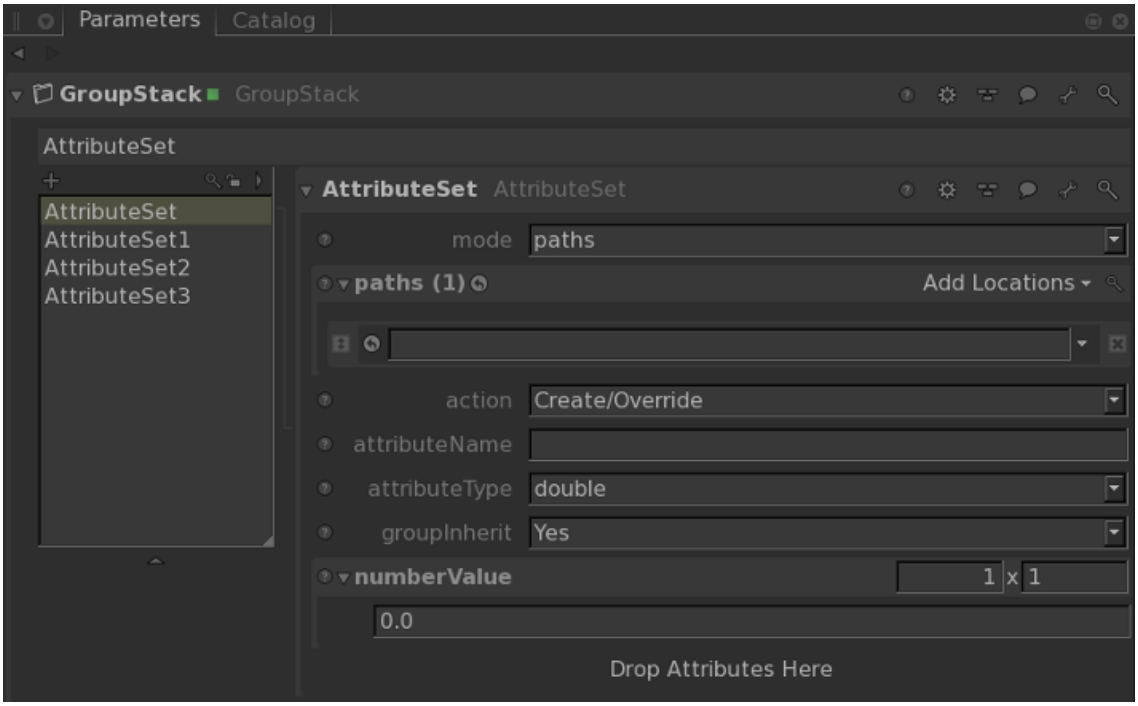
Another way to view the contents of a group is to **Ctrl**+middle-click on the Group in the node graph to drop down a level in the node graph to the level of the child nodes. With the contents of a group exposed, you can select and edit individual nodes in the same way you would any other. See Editing the Node Graph for more on selecting nodes and editing parameters. To go back up a level in the node graph, use **Ctrl**+**Backspace** or click on the up arrow icon in the node graph address bar. 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.

Note: See the Adding User Parameters section for more on adding and using user parameters. Additionally, for more on Group nodes and their incoming or outgoing connections, see **Help** > **Developer Guide**.

GroupStack and GroupMerge Nodes

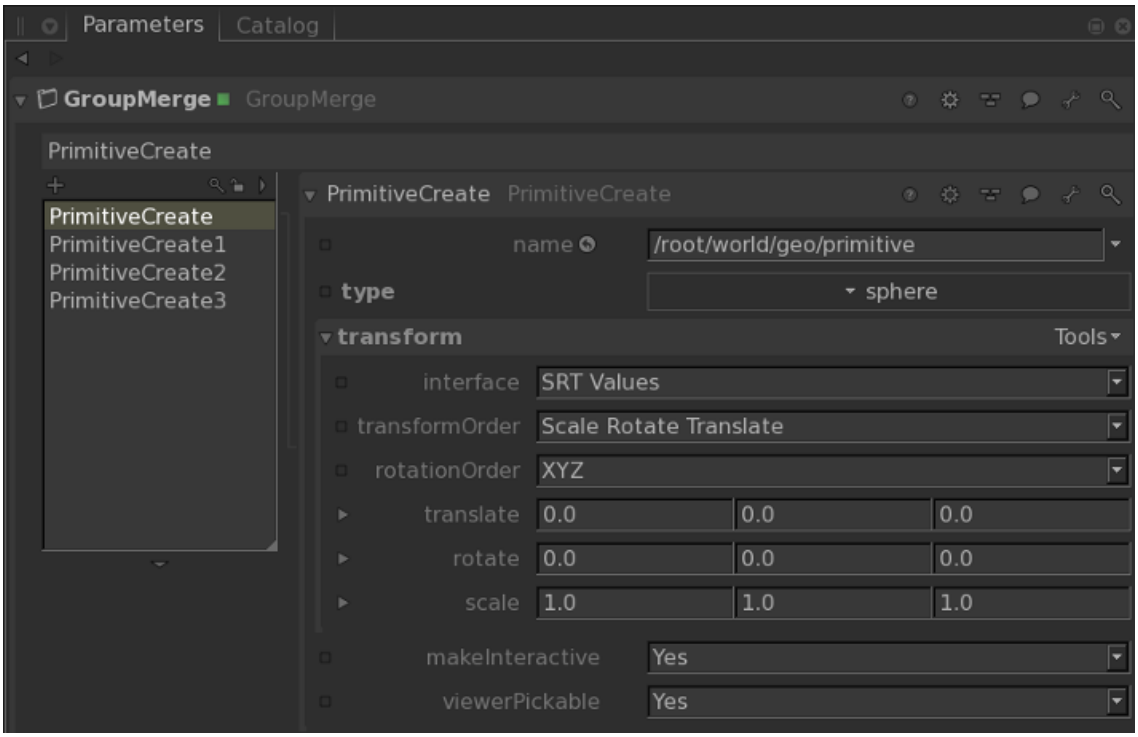
As well as standard Group nodes, which can have child nodes of any node type, there are two special convenience group nodes to make it easier to collect multiple nodes of the same type: GroupStack and GroupMerge. These nodes are useful for organizing the UI when there are large numbers of the same types of nodes. Both GroupStack and GroupMerge nodes offer an interface to control some elements of the nodes contained within. 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.

A GroupStack node can only contain nodes of a single type, and those nodes must have at least one input connection, and a single output connection. The contained nodes are chained together in a series inside the GroupStack, with the output of the top node connected to the first input of the next node, and so on. If there are multiple inputs on a node, only the first one is considered. **Ctrl**+middle-click on the GroupStack node in the node graph to see the arrangement of the nodes inside the GroupStack.



In the node graph, select any nodes you want to place in a GroupStack node, then **Shift**+middle-click and drag them onto the GroupStack node's list pane in the **Parameters** tab. The nodes are transferred to the GroupStack node, and are editable in the GroupStack node's **Parameters** tab. GroupStack nodes have additional **Parameters** tab options for adding, and editing nodes of their selected type. Above the list pane in a GroupStack node's **Parameters** tab are icons to add additional nodes of the selected type, filter the list of nodes, and lock highlighted nodes. Highlighting an entry in the list pane and pressing the **D** key also disables the selected entry.

A GroupMerge node can only contain nodes of a single type and those nodes must have a single output connection, but there is no requirement on the type of input connections used. The child nodes are connected as separate inputs to a Merge node, creating a single output out of the group. Any inputs on nodes inside a GroupMerge node are ignored. The only connections inside the GroupMerge are from the outputs of the constituent nodes, to the input of a Merge node. **Ctrl**+middle-click on a GroupMerge node in the Node Graph to see the arrangement of its constituent nodes.



In the node graph, select any nodes you want to place in a GroupMerge node, then **Shift**+middle-click and drag them onto the GroupMerge node's **Parameters** tab. The nodes are transferred to the GroupMerge node, and are editable in the GroupMerge node's **Parameters** tab. GroupMerge nodes have additional **Parameters** tab options for adding, and editing nodes of their selected type. Above the list pane in a GroupMerge node's **Parameters** tab are icons to add additional nodes of the selected type, filter the list of nodes, and lock highlighted nodes. Highlighting an entry in the list pane and pressing the **D** key disables the selected entry.

Introduction to 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. Macros wrap functionality, allowing you to hide or group logic, so that you can then re-use the macros multiple times. You can create a macro made from groups or from any single node by using the menu at the top of the node's **Parameters** tab, and choosing **Save As Macro...**.

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**. Like any other node, they can make use of user parameters for more complex functionality. For more on how to set up user parameters, refer to Adding User Parameters.

You can also place and 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 a 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 are automatically 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.

For more information on creating or recalling macros, refer to Macros.

Introduction to SuperTools

SuperTools are like macros 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 SuperTools, refer to Writing a SuperTool.

Alternatively, for more technical information about how to set up SuperTools using the NodegraphAPI and the Plug-in Registry, refer to SuperTools.

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 **Tabnode 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/studio** you can put macros in **/production/katana/studio/Macros**. Macros which are inside a subdirectory automatically suffixed with the name of the subdirectory.

For example:

Macro Path	Macro Name
/production/katana/studio/Macros/MyMacro.macro	MyMacro
/production/katana/studio/Macros/_User/MyMacro.macro	MyMacro_User
/production/katana/studio/Macros/OtherMacros/MyMacro.macro	MyMacroOtherMacros

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.

SuperTools

SuperTools are compound nodes where the internal structure of nodes is dynamically created using Python scripting.

This means that the internal nodes can be created and deleted depending on the user's actions, as well as modifications to the connections between nodes and any input or output ports. The UI that a SuperTool shows in the **Parameters** tab can be completely customized using PyQt, including the use of signals and slots to create callbacks based on user actions. To help with this, we have a special arrangement with Riverbank Computing - the creators of PyQt - allowing us to give user access to the same PyQt that Foundry uses internally in Katana.

Many of Katana's common user level nodes, such as the Importomatic, GafferThree, and LookFileManager, are actually SuperTools created out of more atomic nodes. It can be useful to look inside existing SuperTool nodes and macros to get a better understanding of how they work. If you **Ctrl**+middle-click on a suitable node, you open up its internal **Node Graph**.

In general, SuperTools consist of:

- A Python script written using the Katana NodegraphAPI that declares how the SuperTool creates its internal network.
- A Python script using PyQt that declares how the SuperTool creates its UI in the **Parameters** tab.
- A third Python script (typically) for common shared utility functions needed by both the nodes and UI scripts.

For more information, see How to Write a SuperTool in the Katana Developer Guide.

Writing a SuperTool

SuperTools in Katana are written in Python and the suggested convention is that form and function are defined by two separate classes: **xxxNode** and **xxxEditor** respectively (where **xxx** is the tool's name, for example, GafferThree). The Editor class offers a UI to modify the internal network using the API functions, whereas the Node class defines the node itself and the public scripting API.

Most of the information on SuperTools can be found in SuperTools, however, continue for the basics on writing a SuperTool.

Following the suggested convention, the internal file structure of a SuperTool could look something like this:

```
HelloSuperTool

├─ __init__.py
└─ v1
    ├─ __init__.py
    ├─ Editor.py
    ├─ Node.py
    ├─ ScriptActions.py
    └─ Upgrade.py
```

The files in a SuperTool are as follows:

- **Node.py** - the node itself and the public scripting API, which you can test if you get a reference to the node in the **Python** tab.
- **Editor.py** - the Qt4 UI (this is only imported in the interactive GUI, not in batch or scripting modes).
- **ScriptActions.py** - useful functions that are not part of the node API. Since this node is imported by both node and editor, it cannot contain any GUI code.
- **Upgrade.py** - stub for upgrading the node. This allows compatibility with older versions of the node.



Note: The clean separation between the node and the UI is important for being able to script the node in Script mode.

There is no namespacing of nodes inside groups or SuperTools. To reliably get access to nodes with an initial name, you should use expressions such as:

```
getNode('Merge').getName()
```

If the node gets renamed, Katana looks for all getNode('xxx') calls and renames them appropriately.

You can also find example source code, which demonstrates the creation of a SuperTool in:

$KATANA_HOME/plugins/Resources/Examples/SuperTools/.

- PonyStack
- ImageCoordinate
- ExamplePackageSuperTool
- PonyFarm
- AttributeWedge

Image Coordinate for example demonstrates the SuperTool making use of a custom Qt widget, and interacting with Katana's Model-View-Controller system. It also presents a simple SuperTool that allows you to load an image (**.jpg** or **.gif**) and select a coordinate within the image. The coordinate is then fed back into the scene graph using the SuperTool's internal node network.

For more information, see How to Write a SuperTool in the Katana Developer Guide.

Customizing GafferThree

Creating a Custom GafferThree Package Class

Package classes in the GafferThree define the types of items that can be created through the GafferThree, the way in which they are displayed in the GafferThree's UI within the **Parameters** tab, and other properties such as whether the items can accept other items as children.

In order to create a custom package class for the GafferThree, the following components are required:

- Package class
- Edit package class (optional)
- UI delegate class
- Package initialization file

Package Class

The package's main class is responsible for creating nodes that produce the scene graph locations and attributes for your package, as well as the parameters for modifying these locations and attributes.

To implement a package class, do the following:

- Create a class derived from **PackageSuperToolAPI.Packages.Package**, or choose a specific type of package class to derive from if appropriate, for example, the GafferThree's **LightPackage** class.
- Implement the **create()** class method. Your implementation should create and connect together the nodes for the package within a Group node, and instantiate and return a new package class instance.
- After defining your package class, register it with the GafferThreeAPI by calling **GafferThreeAPI.RegisterPackageClass()**, and passing your class.

Tip: You can store references to nodes you create using the **PackageSuperToolAPI.NodeUtils.AddNodeRef()** function. **AddNodeRef()** stores the name of the node as a custom parameter on the package node, and makes it easy to refer to nodes within a package from elsewhere in the code associated with your package. Node references stored in this way can be obtained using the corresponding **GetNodeRef()** function from the **NodeUtils** module.

Edit Package Class (Optional)

An edit package class is responsible for creating nodes, which can edit scene graph locations in the incoming scene graph. These locations may have been created by an instance of your custom package class in an upstream GafferThree node.

To implement an edit package class, which is optional, do the following:
Create a class derived from **PackageSuperToolAPI.Packages.EditPackage**, or choose a specific type of edit package class to derive from if appropriate. For instance, the GafferThree's **LightEditPackage** class.

UI Delegate Class

A UI delegate class is responsible for defining the parameter interface shown in tabs below the Gaffer object table in the **Parameters** tab.

To implement a UI delegate class, do the following:

- Create a class derived from **PackageSuperToolAPI.UIDelegate.UIDelegate**, or choose a specific type of UI delegate class to derive from if appropriate. For instance, the GafferThree's **LightUIDelegate** class.

Tip: You can use **PackageSuperToolAPI.UIDelegate.GetUIDelegateClassForPackageClass()** to obtain the UI delegate class that was registered for a specific package class.

- You can also implement the **getTabPolicy()** instance method, which is optional:

getTabPolicy() receives the name of a tab, **Object**, **Material**, or **Linking** in the case of GafferThree, and is expected to return a **QT4FormWidgets.PythonGroupPolicy** instance containing parameter policies for editing parameters on the nodes created by your package class.

Tip: You can use **PackageSuperToolAPI.NodeUtils.GetRefNode()** to reference nodes in your package that you have previously stored using **AddNodeRef()**.

Package Initialization File

To ensure that your package is initialized, place your package modules in a **SuperTools** sub-directory of a path, which is contained in your **$KATANA_RESOURCES** environment variable, alongside an **__init__.py** file, which imports your package files.

The resulting directory structure should look similar to the following:

```
SuperTools
`-- SkyDome
    |-- __init__.py
    |-- ExamplePackage.py
    `-- ExampleUIDelegate.py
```

The UI delegate class can only be imported if Katana is running in UI mode:

```
import PackageSuperToolAPI
import ExamplePackage

if PackageSuperToolAPI.IsUIMode():
    import ExampleUIDelegate
```

Optimizing Performance

Katana node graphs, their Op trees, and the scenes they subsequently create are incredibly flexible and varied. To optimally evaluate these scenes, an evaluation engine must be both efficient and flexible enough to handle the variety and complexity of scenes possible in Katana.

Geolib3-MT, the next generation of Katana scene graph processing engine, provides a greater degree of configuration, introspection, and tuning options than previous versions of Geolib3 to meet the demands of increasingly complex and varied workloads.

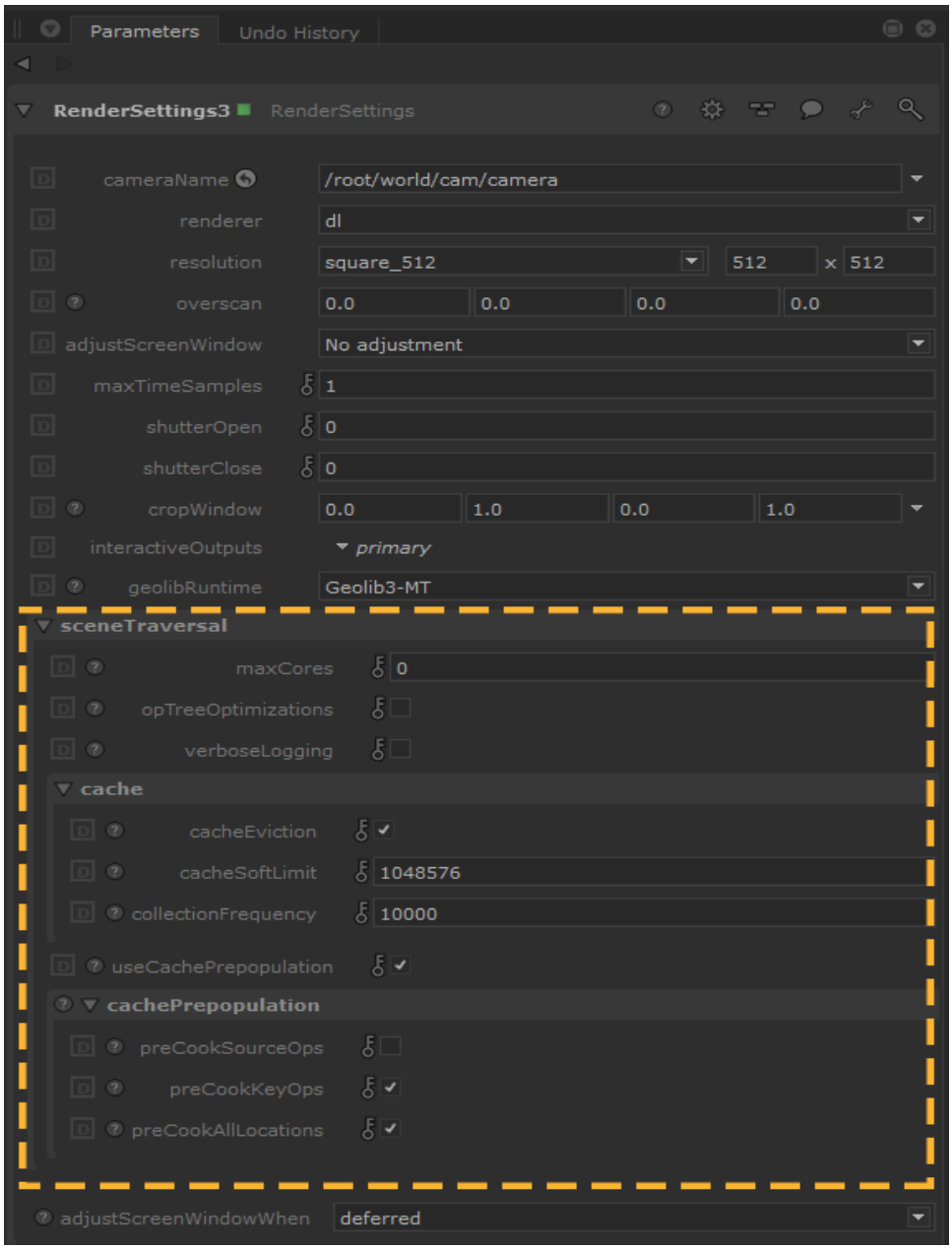
This section explores these options and how they can be used to improve scene traversal performance.



Note: For more information on profiling and optimization, you can refer to the Developer Guide.

Geolib3-MT Configuration

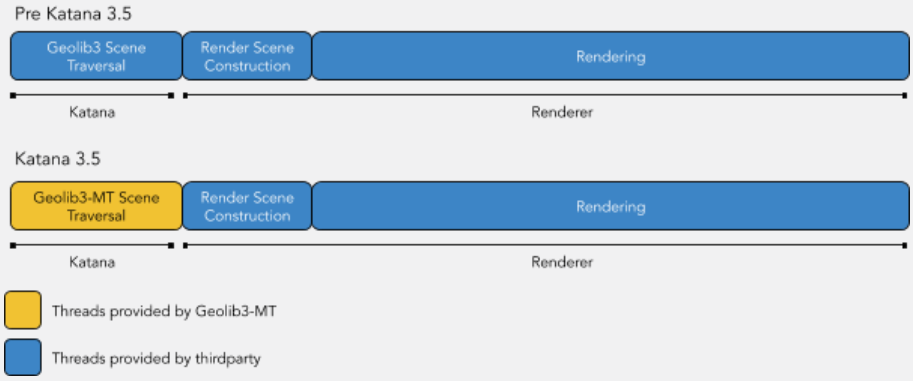
Geolib3-MT can be configured via the **RenderSettings** node. All Geolib3-MT options live under the **sceneTraversal** heading.



sceneTraversal Parameters

maxCores *text field* Default: 0	Determines how many logical cores Geolib3-MT uses during scene traversal phase. Unlike previous versions, Geolib3-MT uses an internal thread pool to improve scene traversal time. The following diagram demonstrates the difference

between Geolib3-MT and previous versions of Katana.



The default value of 0, causes Geolib3-MT to use all available logical cores on the host computer.

Note: Whilst the core Geolib3-MT processing engine scales as the number of cores increases, individual Ops within an Op tree may not exhibit the same scaling characteristics. It is possible that an increase in threads can result in an increase of scene traversal time. In this case, the new profiling tools available in Katana 3.5 can be used to identify these Ops and optimize their behavior. The same is true of Ops marked **thread unsafe**, as these require the acquisition of a **Global Execution Lock** (**GEL**), which further limits scene traversal scalability.

opTreeOptimizations

checkbox

Default: Off

When turned on, Geolib3-MT performs a pre-processing step in which it examines the topology of the Op tree to identify constructs that can be potentially optimized. One optimization is the collapsing of sequences of Ops of the same type into a single instance of that Op. There are a number of benefits to this:

- Reduced function call overhead - There is a small cost involved in scheduling an Op to cook a scene graph location. By combining chains of similar Ops, it's possible to reduce this function call overhead.
- Reduced memory footprint - A chain of 10 Ops occupies 10 separate cook results in the caching subsystem, while a successfully collapsed Op Chain occupies only 1 cook result per location.

The result of evaluating a collapsed chain of Ops, when observed from the most downstream Op, should be the same as if the chain of Ops were evaluated.

	 **Note:** Op API calls to query upstream scene graph results, such as **getAttr()**, does not return the expected result when called within a collapsed chain if one of those Ops within the chain was responsible for setting that attribute. In this case the Op should use **getOutputAttr()** instead. The Op tree optimizer attempts to collapse any chain of Ops of the same type if it calls **GeolibSetupInterface::setOpsCollapsible()** during the **setup()** call. Callers of this function must specify the name of an attribute which Geolib3 passes to the Op's **cook()** call as an Op argument. This attribute contains an ordered array of attributes (ordered upstream Op to downstream Op) containing the collapsed Ops' arguments. The Op is then able to deal with this batch Op argument as appropriate.
verboseLogging *checkbox* Default: Off	When turned on, verbose logging inside the Geolib3-MT runtime is enabled. This includes verbose logging during scene traversal, and Op tree optimization, if enabled.

sceneTraversal.cache Parameters

Geolib3-MT includes a number of settings to control the behavior of the caching subsystem. The caching subsystem is responsible for the storage and retrieval of previously cooked scene graph locations, known as **cook results**. These settings can be modified from the **RenderSettings** node on a project-by-project basis.

Caching, and the trade-off between memory usage and time to first pixel can have a significant impact on the performance of scene traversal time and rendering. Using the settings provided by Geolib3-MT it's possible to tune the memory footprint during the scene traversal phase of rendering.

cacheEviction *checkbox* Default: On	If turned off, no cook results are evicted from the cache.

	Tip: Whilst initially it might seem counter-intuitive to disable cache eviction, there may be scenes where it is appropriate. This may be the case when the scene and data structures required by the renderer fit comfortably into memory. Even larger scenes could benefit to some degree, as once the scene generation phase of rendering is complete, the memory pages occupied by Geolib3-MT's cook results can no longer be accessed and therefore are not eligible for paging to disk; as these pages won't be re-paged to main memory during rendering the performance penalty is minimal.
cacheSoftLimit *text field* Default: 1,048,576	If **cacheEviction** is turned on, the **cacheSoftLimit** governs how many cook results are stored in local caches before entries are evicted using a least recently used eviction policy. **Note:** Whilst these entries may be evicted from a local cache they may be shared amongst a number of other local caches or the central (shared cache). In which case, the entries' memory won't immediately be reclaimed. Consider the maximum depth of the scene graph and Op tree. The **cacheSoftLimit** controls the size of the recently used cook result cache on a per thread basis. This means any locations cooked on a particular thread, or any locations accessed during the cooking process (such as via **getAttr()**), are stored in the local cache and subject to eviction based on the value of the **cacheSoftLimit**.
collectionFrequency *text field* Default: 10,000	If **cacheEviction** is turned on, the **collectionFrequency** governs the time, in milliseconds, between collection cycles. During a collection cycle, Geolib3-MT gathers all cache entries evicted since the previous collection cycle and if the cook result is no longer used, evict and reclaim the memory for the cook result. **Note:** Reducing the **collectionFrequency** interval causes more aggressive eviction of cook results leading to a reduced memory footprint but potentially at the cost of scene traversal time.

useCachePrepopulation *checkbox* Default: On	If turned on, Geolib3-MT performs a traversal of the scene graph populating an internal cache. The extent of this traversal can be controlled by the settings under **sceneTraversal.cachePrepopulation**.

sceneTraversal.cachePrepopulation Parameters

preCookSourceOps *checkbox* Default: Off	If turned on, Geolib3-MT first fully traverses the scene generated by any source Op (any Op with no inputs) found in the Op tree. This setting can provide benefits when loading in geometry caches or other asset types.  **Note:** Empirical tests have found that source Ops are typically followed by some form of prune operation; as a result, in these cases, turning on **preCookSourceOps** can generate more scene graph locations than is required which can lead to increased memory consumption and traversal times.
preCookKeyOps *checkbox* Default: On	If turned on, Geolib3-MT identifies Ops within the Op tree that can be evaluated in parallel. An example of this is the **Merge** Op: Geolib3-MT evaluates each branch in parallel, which can reduce scene traversal times.
preCookAllLocations *checkbox* Default: On	If turned on, Geolib3-MT cooks all remaining scene graph locations, fully expanding the scene.

Based on the values of the above settings, on completion of the **cachePrepopulation** phase, the Geolib3-MT cache is pre-populated with either the whole scene graph or a subsection of it. Geolib3-MT has been optimized to provide efficient access to renderer plugins via the existing **FnScenegraphIterator** API to this cache. This cache is a scalable, thread safe cache, as such >we encourage renderer plugins to access this cache concurrently to improve the performance of the scene build phase.



Warning: If the Geolib3-MT cache is not fully populated, cache access (via **FnScenegraphIterator**) results in a cache miss. In this case the requested location is cooked using the calling thread.

Geolib3-MT Profiling

Geolib3-MT adds a new render type, called **Preview Render with Profiling**, designed to help track down performance problems in scene traversal. This performs a normal **Preview Render**, but also captures information about which Ops have run, the amount of CPU used by them to cook locations, and the amount of memory used for attributes and Lua scripts.

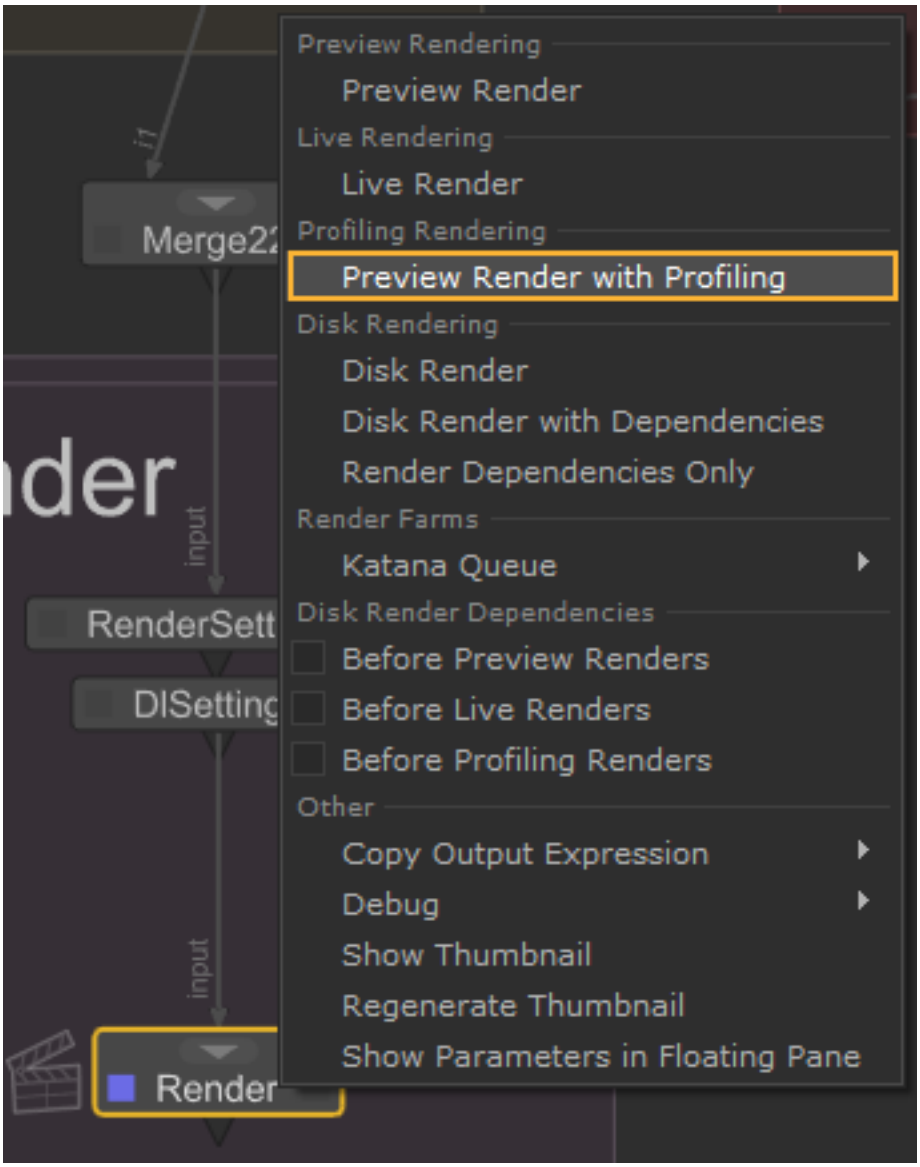
A Preview Render with Profiling outputs profiling data in two places:

- A summary report in the **Render Log**, containing total CPU time and memory used, as well as the ten most expensive Ops.
- A JSON file written to disk containing raw profiling data.

Starting a Preview Render with Profiling

A **Preview Render with Profiling** can be started from the same menu as any other render:

1. Right-click on the node you wish to render from.
2. Click **Preview Render with Profiling** from the menu.



This option will be available for any renderer that already supports a **Preview Render**, and requires no additional work on the part of the renderer. If the renderer implements the **finalize()** method of the Geolib3-MT runtime, these profiling reports will be created when the runtime is finalized; otherwise reports will be written when the render finishes.

The overhead of capturing profiling data during a **Preview Render with Profiling** is minimal, and there should not be significant slowdown compared to a normal **Preview Render**.

What Information is Captured?

A **Preview Render with Profiling** captures the following information for each Op that is executed during scene traversal:

- The **name**, **type** and **numerical ID** of the Op.
  Each Op has name, type and a unique numeric ID. For example, an **OpScript** Op may have name **op74**, type **OpScript.Lua**; and ID **77**.

  Note: The **name** and **ID** do not need to correlate.

- The **name** and **type** of the Katana node that spawned the Op.
  In cases where an Op is spawned directly by a Katana node, the **name** and **type** of that node are recorded. In cases where the Op was created implicitly, the node name will equal **_NoName_** and the type will equal **_NoType_**. This occurs, for example, with **MaterialFilenameResolve** Ops as these Ops are created implicitly when a file name needs to be resolved, so no Katana node is identified as the creator.

  Note: If **sceneTraversal.opTreeOptimizations** is enabled and chains of Ops are collapsed, node **name** and **type** will be replaced with a string generated from the chain. If the chain has length **t**, formed of Ops of type **opType**, where **Op k** is named $\mathbf{o_k}$ and is generated by a Katana node named $\mathbf{n_k}$, the general form of the string will be:
  $\mathbf{cop(o_0(n_0)->o_1(n_1)->\ ...\ ->o_t(n_t))}$
  However, the format of this string is not guaranteed to remain fixed.

- The **total CPU time** that Op spent cooking locations.
  Each Op will cook many locations, and the time spent doing this, across all scene traversal threads, is accumulated. CPU time scales with the number of scene traversal threads when a scene is traversed in parallel. If this is not the case, there may be a thread-unsafe Op upstream of the Op in question.
- The **memory footprint** of that Op.
  Each Op must allocate memory to cook locations, and the memory total per Op is aggregated.
  At present, the following allocations are recorded:

  - Allocations made by the **FnAttribute** library, to store attributes of cooked locations.
  - Allocations made by the **Lua** interpreter while executing OpScripts.
  - Allocations made to store **CookResults** in the cache.

Profiling Summary Report

A summary report is written to the **Render Log** upon completion of a **Preview Render with Profiling**. This report is intended to give a high-level overview of the profile data, and contains:

- The total CPU time, summed across all Ops.
- The total memory footprint, summed across all Ops.

- The slowest five Ops by CPU time.

```
============================================================
Profiling summary:
  Total CPU time:    1m37.927s
  Total memory used: 16.095 GiB
  Total duration:    18.468s

Most costly Nodes:
  CPU Time       | Node Name                      | Node Type
     68802.488ms | _NoName_                       | _NoType_
     19461.750ms | LookFileResolve5               | LookFileResolve
      3239.540ms | LookFileLights_Resolve         | LookFileResolve
      2394.819ms | ScenegraphXml_In               | ScenegraphXml_In
      1729.306ms | restoreLightsDefaultValues4    | GenericOp

Most costly Ops:
  CPU Time       | Op Type                        | Node Name                      | Node Type
     34354.827ms | OpResolve                      | _NoName_                       | _NoType_
     17834.980ms | LookFileResolve                | LookFileResolve5               | LookFileResolve
      9322.987ms | MaterialResolve                | _NoName_                       | _NoType_
      3564.734ms | ReferentialInheritanceResolve  | _NoName_                       | _NoType_
      3071.861ms | ConstraintResolve              | _NoName_                       | _NoType_

Runtime profiling file written to: /tmp/katana_tmpdir_27518/profile_profilingMockRenderer_previewRender_2020-02-04T14-28-47Z.json
============================================================
```

The relevant section of an example **Render Log.**

Profiling JSON File

In addition to the summary report, a JSON file containing the raw profiling data is written to disk. The directory it is written to is determined by the **--profiling-dir** command-line argument; if this is not set, it will be written to the temporary directory for the Katana session. If this directory does not exist, it will be created, as long as file system permissions allow. The file name takes the following format:

`profile_<renderer>_previewRender_<datetime>.json`

Where:

- <renderer> is the name of the render plugin, e.g. dl for 3Delight;
- <datetime> is the ISO8601 timestamp from when the render was started.

The file contains a single JSON object with the following properties:

Property	Type	Description	Example
timestamp	*string*	ISO8601 timestamp at which the profile file was written.	2019-10-11T09:37:06Z
renderer	*string*	Name of the render plugin.	dl
renderMethodName	*string*	Name of the render method; currently always **previewRender**.	previewRender
environment	*object*	An object containing values of	{

		various environment variables, including: • KATANA_RELEASE • KATANA_ROOT • KATANA_RESOURCES.	"KATANA_RELEASE": "3.5v1", "KATANA_ROOT": /opt/foundry/katana3.5v1", "KATANA_RESOURCES": "<unset>" }
profileMode	*string*	Name of the profile mode; currently always **basic**.	basic
ops	*array*	Array of objects describing resources consumed by each Op.	See the table below
numOps	*number*	Length of the Ops array.	78
wallTime	*number*	Wall-clock time in seconds between render start and the profiling file being written; if the renderer implements **finalize()**, this equates to scene traversal time.	46.85064
cpuTime	*number*	Sum of CPU time for all Ops, in seconds.	91.39238
memoryUsed	*number*	Sum of memory footprints for all Ops, in bytes.	10728607911

The ops property contains an array of objects of the following format, one for each Op that was executed during scene traversal.

Property	Type	Description	Example
opId	*number*	The unique integer identifier for the Op.	23
opName	*string*	The unique name of the Op.	op223
opType	*string*	The type of the Op.	AttributeSet
nodeName	*string*	The name of the Katana node responsible for creating this Op, or **_NoName_** if the Op was created implicitly.	RenderSettings_SetSamples

nodeType	*string*	The type of the Katana node responsible for creating this Op, or **_NoType_** if the Op was created implicitly.	RenderSettings
cpuTime	*number*	The total time this Op spent cooking locations across all threads, in seconds.	0.54512136
memoryUsed	*number*	The total memory footprint, as defined above, this Op used while cooking locations, in bytes.	185378321

Analyzing the Profile Results

A Python 2.7 script is included to sort and group the results in various ways. This script can be found here:

```
$KATANA_ROOT/extras/Profiling/analyzeProfilingRenderResults.py
```

You can call this Python script from the command line as follows:

```
cd $KATANA_ROOT/extras/Profiling
python analyzeProfilingRenderResults.py /path/to/results/file.json <options>
```

The following command-line options are available:

--help	Display the help text and exit.
--sort-by FIELDNAME	Sort the results by FIELDNAME, where FIELDNAME is one of the JSON property names opId, opName, opType, nodeName, nodeType, cpuTime or memoryUsed.
--reverse, -r	Sort the results in reverse order.
--group-by FIELDNAME	Group the results by FIELDNAME, where FIELDNAME is one of the JSON property names opId, opName, opType, nodeName, nodeType, cpuTime or memoryUsed.
--human-readable, -h	Print memory totals in human-readable units (i.e. KiB, MiB, etc. as appropriate) rather than bytes.
--limit LIMIT, -l LIMIT	Limit output to the first LIMIT rows after grouping and sorting.
--columns COLUMNS	Output only the specified columns, where COLUMNS is a comma-separated list of the JSON property names opId, opName, opType, nodeName, nodeType, cpuTime or memoryUsed.

The script outputs an ASCII table of results to **stdout**, grouped and sorted as requested. If **--sort-by** is not set, results will be sorted by **opId**. If **--group-by** is not set, no grouping will occur.



Note: When grouping by **nodeName**, all results with name **_NoName_** will be grouped together. The same is true for **nodeType**.

The following combination of command-line options may be useful to get started:

--group-by opType --sort-by cpuTime	Find out which Op types are most CPU-intensive.
--group-by nodeName --sort-by cpuTime	Find out which Katana nodes are most CPU-intensive.
--group-by nodeType --sort-by memoryUsed	Find out which Katana node type accounts for the largest memory footprint.

Op Cook Profiling

A Profiling mode exists in Katana. When launched in Profiling mode, Katana can generate reports on Op cook times within profiling sessions. Cook times are aggregated per Op instance, per location. These reports can help in pinpointing slow areas of the Op tree and scene graph.

The profiling is low overhead to minimize its effect on cook performance and the more computationally demanding data aggregation is performed at the end of the profiling session.



Note: The timing report files produced can be extremely large, of the order of gigabytes. As a guide, note that typically one line of information is produced per Op instance, per scene graph location at which it is cooked. This is why we recommend you use profiling mode selectively when measuring cooking times, and not during normal production usage of Katana.

Command-Line Options

The following command-line options allow you to set the specific Profiling mode you want to use, or to specify a directory where profiling reports are written:

• **--profile** - launches Katana in Profiling mode,

- **--force-profile** - launches Katana in Profiling mode and starts a profiling session immediately on launch,
- **--profiling-dir=[DIR]** - specifies the directory where the profiling reports are written.



Note: For more information on command-line launch modes in Katana, see Command-line Interface

Starting and Ending a Profiling Session

Learn the basics of beginning and ending a profiling session in Katana.

Profiling Renders

An overview for profiling renders in Katana.

Profiling Reports

A guide for creating and analyzing profiling reports.

Starting and Ending a Profiling Session

A profiling session can be started and ended on Katana's main Runtime using the following Python calls:

```
from Katana import FnGeolib

#Get the Runtime
runtime = FnGeolib.GetRegisteredRuntimeInstance()

#Start Profiling
runtime.StartProfilingSession()

...

#End Profiling
runtime.EndProfilingSession()
```

This code can be called from a plug-in, a shelf script, or in the Python console in the standard interactive mode. It can also be invoked in **--script** and **--shell** modes. It is possible to start a profiling session on Katana startup and end it when Katana quits by using the **--force-profile** command-line option:
`./katana --force-profile`

Note: This option causes all Geolib3 Runtime instances to start a profiling session upon initialization, meaning that reports are also created for additional instances, such as that used to cook Viewer proxies.

Note: Since it is more contrived for you to start and end a profiling session, in **--batch** mode, Katana immediately starts a profiling session when launched with either the **--profile** or **--force-profile** options, and it ends the session once the render is finished. These options are, therefore, equivalent in **--batch** mode.

Profiling Renders

Renders are executed by the **renderboot** sub-process launched by Katana's **katanaBin** main process. This means that render processes have their own instances of the Geolib3 Runtime that can be profiled as well. Any renders launched in the standard, Interactive Katana mode, as well as the Shell and Script modes during a profiling session (after **StartProfilingSession()** and before **EndProfilingSession()** on Katana's main Runtime) are profiled and produce their own profiling reports at the end. In Batch mode, the render is profiled if the **--force-profile** or **--profile** options are used.

Profiling Reports

Creating Profiling Reports

Profiling reports are generated at the end of each profiling session per process (**katanaBin** versus **renderboot**, for example), per Runtime instance. Renders that call the Katana procedural on multiple threads, for example, instantiate one Runtime per thread, each one producing a different report. Each report consists of two files:

1. A **.dot** file that contains the graph with the Op tree at the end of the profiling session.
2. A **.csv** file containing the recorded cooking times in a table format.

The profile reports are, by default, written to the Katana session's temporary directory and, optionally, in the directory specified by the **--profiling-dir** command-line option. For example:

```
./katana --profile --profiling-dir=/tmp/katana_profiling
```

The naming of the reports follows the format:

```
Profiling__PID[P]__Runtime[R]__[YYYY]-[MM]-[DD]__[hh]-[mm]-[ss].csv
Profiling__PID[P]__Runtime[R]__[YYYY]-[MM]-[DD]__[hh]-[mm]-[ss].dot
```

In which **P** is the Process ID of the process that produced the report and **R** is a number that identifies the instance of the Runtime. In a multi-threaded render that instantiates several Runtimes, there is a **P** for **katanaBin** and another for **renderboot** and, in the latter, one **R** per Runtime instance created at render time. The date and time at which the profiling session was ended is appended to the file name.

The **.dot** files can be converted, for example, into a PDF document using the following command, which requires Graphviz:

```
dot [DOT FILE] -Tpdf > [OUTPUT PDF FILE]
```

Analyzing Profiling Reports

The **.csv** (comma-separated values) files contain the aggregated cook times and number of cooks. This can be read directly into a spreadsheet application or other reporting tool. Each entry (row) contains the following values (columns):

1. **OpId** - integer ID of the Op instance (or the invoking Op instance if **IsExecOp** is **true**).
2. **OpType** - Op type string (for example, "AttributeSet").
3. **IsExecOp** - specifies whether or not the entry refers to an Op invoked using **execOp()** (**true** or **false**).
4. **Location** - the path of the scene graph location that was cooked
5. **TotalTime(usecs)** - the total time spent by the Op instance in successfully cooking **Location** (in microseconds).
6. **AbortTime(usecs)** - the total time spent by the Op instance in aborted cooks of **Location** (in microseconds).
7. **TotalCount** - the number of cooks of the Op at **Location**.
8. **AbortCount** - the number of aborted cooks of the Op at **Location**.

An entry for which **IsExecOp** is **true** represents the times for an Op that was explicitly cooked by an invoking Op, using a call to **execOp()**. In this case, the **OpId** corresponds to the invoking Op instance, while the **opType** corresponds to the type of the invoked Op.

Note: Time reported for entries for which **IsExecOp** is **true** is also included in the entry for the invoking Op. The total cook time during the session is therefore the sum of the cook time for all entries for which **IsExecOp** is **false**.

For example, if Op A (for which **IsExecOp** is **false**) calls **execOp()** on Op B, which in turn calls **execOp()** on Op C, then **TotalTime** for Op A is strictly greater than that for Op B, which is strictly greater than that for Op C. These Ops are all reported with the same **OpID**: that of Op A.

Note: Abort time is a normal component of scene expansion in Geolib3, but is minimized through good Op-writing practices. For more information, see The Op API.

Currently, the Geolib3 Runtime has no knowledge of the mapping between Ops and their respective project nodes. However, the graph produced by the .dot file shows the Runtime's Op tree structure at the time the profiling session was ended, and can assist in matching Ops to nodes. Note that profiling the cooking of distinct or altered Op trees within a single session is likely to produce less helpful and, perhaps, invalid results, since aggregation may erroneously occur across re-purposed Op instances.

The Op tree is created directly from the node graph, plus all implicit resolvers, Interactive Render Filters, and terminal Ops on different UI elements. Note that some nodes produce multiple Ops. **OpId** values in the graph, of course, correspond to those in the respective **.csv** file.

When loading the **.csv** file in a spreadsheet, the values can easily be filtered, sorted, aggregated, and summed. Average times can be calculated by dividing the times by the number of cooks. It is quite possible to produce extremely large reports (to the order to millions of entries) that may not be loaded completely into common spreadsheet software. For such cases, the **.csv** format is readily parsed by a script or program that aggregates values, for example, per location or per Op type.

Profiling and Optimization Guide

This guide describes a number of practical ways in which you can improve the performance of your Katana scenes and reduce render time. It also covers a number of systematic methods by which you can analyze your Katana scene to limit the time you need to spend optimizing your scene.

At times, some of these recommendations may appear contradictory. This is intentional: scene graphs, their Ops and inputs vary significantly between projects. What works for one scene may cause a slow down in others. Profiling and optimization should be an iterative, results driven process; with experience, you will develop an intuition as to what works well in certain situations and for certain scenes.

A general workflow pattern for optimizing Katana scene traversal is as follows:

1. Identify an **optimization target**, for example, time to first pixel, memory reduction in **cook()** calls.
2. Identify thread unsafe ops and refactor where possible to allow parallel evaluation.
   Measure against (1).
3. Identify the most costly ops in the op tree.
   - Analyze these ops and, where possible, optimize the code.
     Measure against (1).
   - Repeat for as many ops as practicable.
4. Analyze data dependencies in the scene graph to exploit op tree parallelism.
   - Refactor node graph where appropriate.
5. Tune cache settings.

Improving Your Node Graph

Understand the various ways of improving your node graph in Katana and learn which to use.

Ways to Improve Your Ops

There are many different ways of improving your Ops in Katana. Learn which to consider and use in a variety of circumstances.

Composing Concurrency-Friendly Scenes

Learn to identify bottlenecks, indirect use of mutexes and other synchronization primitives to understand Katana scene throughput.

Improving OpScript Performance

This guide takes you through the ways to improve OpScript performance by highlighting key areas of knowledge which are essential for scene optimization.

Optimize Projects Using the Performance Tab

Efficient projects are extremely important in production, the less time you spend investigating issues, the more time you can spend on look development or lighting. The **Performance** tab helps you understand where bottlenecks may lie in your node graph by interpreting profiling information.

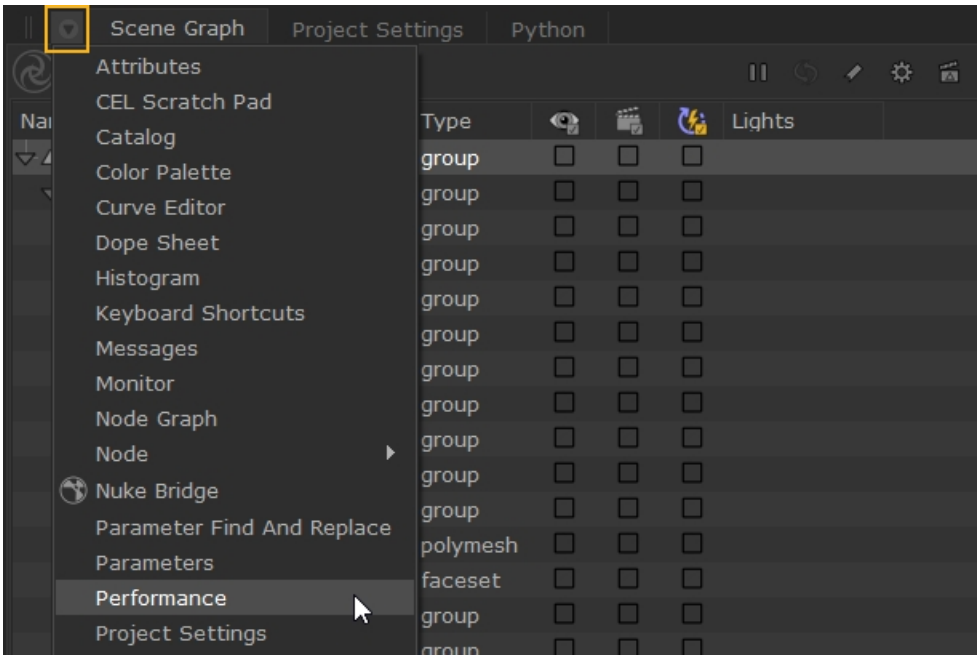
You can see at glance how many nodes you have in the recipe, how many are contributing to your scene, and which nodes use more CPU resources. The Heat Map allows you to view this information directly in the Node Graph tab, providing a visual indication of expensive nodes so you can debug scenes more effectively.

If you're still not sure where optimizations can improve your scene, you can send the Profile Results File to a colleague for analysis. Profiling data is automatically saved as .json files to the KATANA_TMPDIR location.

Tip: The **.json** location is also recorded in the Render Log if you're not sure where the file is saved. See Renderer Logging for more information.

Analyze the Scene with the Performance Tab

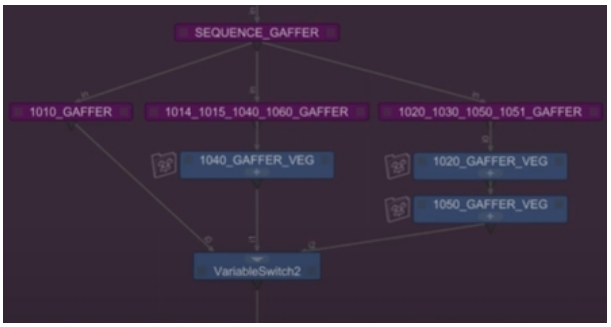
1. Load the project you want to analyze and optimize.
2. Navigate to the **Performance** tab or click the Add Tab button and then select **Performance** if the tab is not already in your workspace.



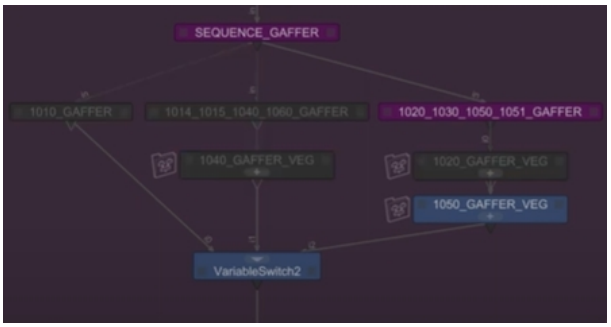
The **Performance** tab is added to the pane and the **Project Stats** displays the number of nodes in the scene and how many of those nodes are contributing.

The **Number of Nodes** and **Contributing Nodes** listed includes all nodes inside groups, Network Material contexts, and super tools such as Gaffer3. As a result, the number may be higher than looking at the node graph suggests.

To see a visual representation of contributing nodes, go to **View** > **Dim Nodes Not Contributing to Viewed Node**. Nodes that are not contributing are dimmed.



All nodes appear to be contributing

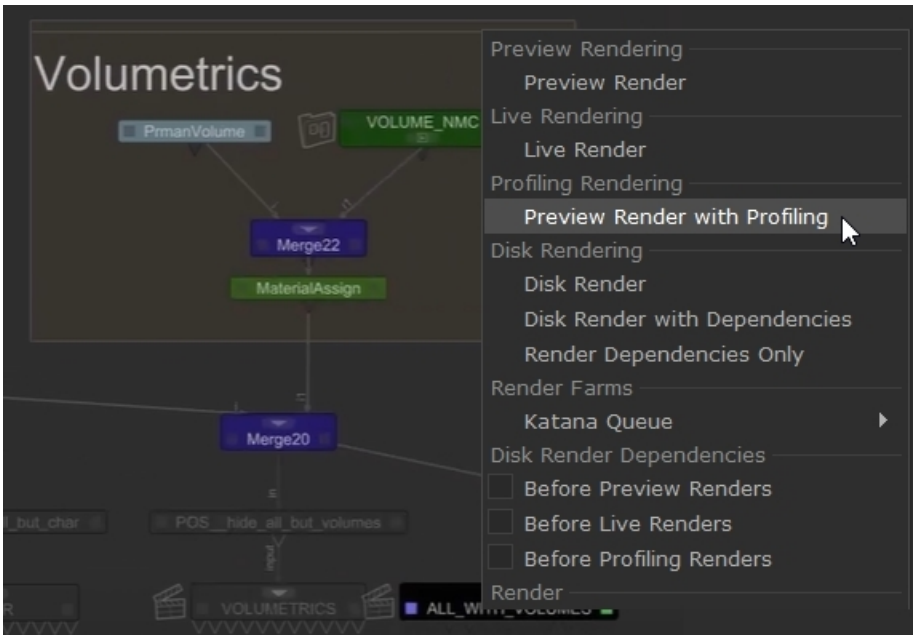


Non-contributing nodes are dimmed

3. Select and deselect nodes in the node graph to update the **Number of Selected Nodes** indicator. The number in brackets indicates nodes that are not visible at the root level, such as those in groups and super tools.

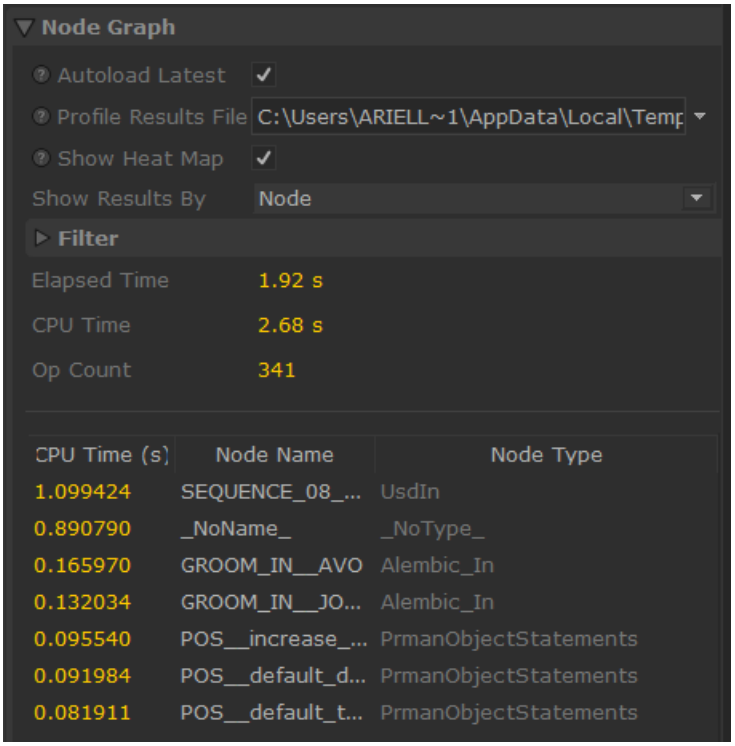
To display more in-depth data

1. Render the scene from any node by right-clicking the node and selecting **Preview Render with Profiling**.



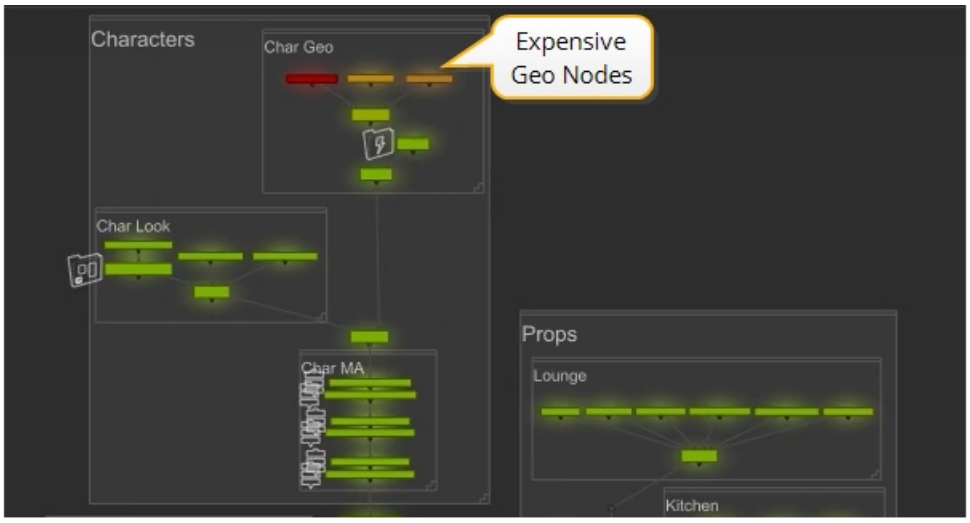
2. The **Performance** tab shows the time to first pixel rendered and a list of nodes and CPU usage, with the most expensive nodes at the top. As you can see in this example, importing geometry (UsdIn) and merging (Merge) are quite CPU-heavy operations.

Note: The **_NoName_** entry represents all operations that are not related to nodes, such as implicit resolvers. See Implicit Resolvers for more information.



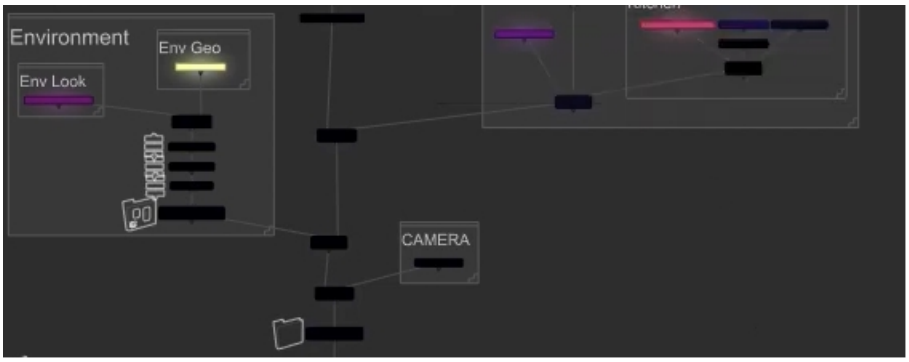
3. Enable **Show Heat Map** to display heat colors in the **Node Graph** tab.

   In this example, the **Char Geo** import nodes are clearly the most expensive nodes.



4. If you have difficulty distinguishing between nodes with similar color gradients, you can change the heat map color scheme by clicking **View** > **Heat Color Map** and selecting a new scheme. For example, **Magma** colors cheap nodes black, which can be easily ignored.

> **Tip:** There's also colored scale at the bottom-left of the **Node Graph** tab that displays color against a relative CPU time value. The range adjusts dynamically between the least expensive node and most expensive node, which allows it to scale to any size of project.



Utilizing the Filtering Options

Once you start using the **Performance** tab more extensively, you might need to start filtering the results seen. As important it is to know which nodes are the most expensive, you might want to hide away certain names or types.

One example of why you might want to do this, is when specific nodes may naturally have a high **cook time** result. For example, nodes which are referencing geometry (e.g. the **UsdIn**) may display the highest **CPU Time** in the list, but it's sometimes not an area that will require any optimization. Narrowing down the search to focus on areas that will need that attention is therefore more useful when optimizing Katana projects.

Options for filtering can be found in the **Performance** tab under **Node graph > Filter**.

• **Show Results** - adjusts the number of results being shown in the list, by default this will be set to 20

• **Node Name** and **Node Type** pages will be available, as well as **Op Name** and **Op Type** pages (when **Show Results By** is set to **Ops**) - all of which include two entries:

• **Include Only** - Will take in a list to only include results that match the conditions specified

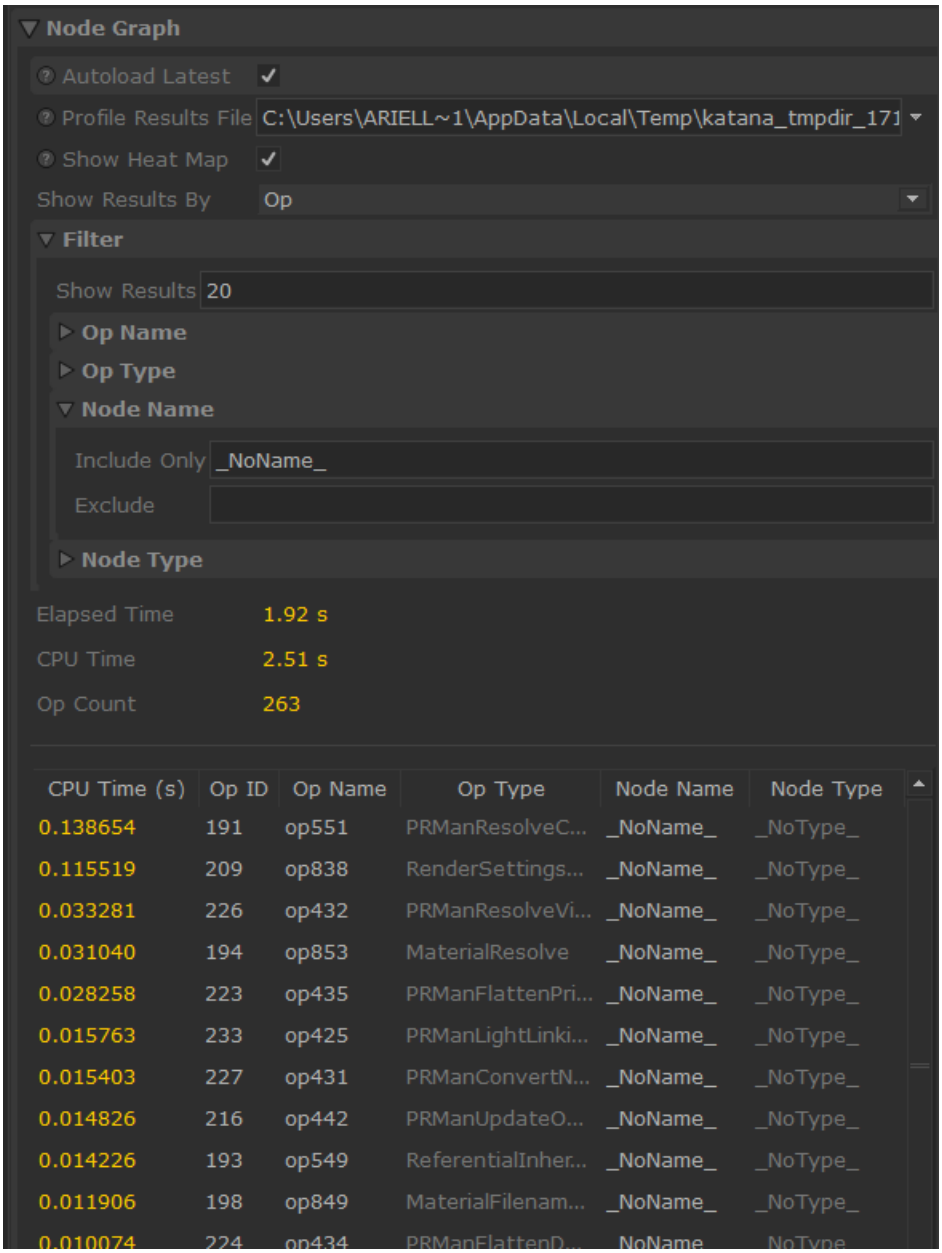
• **Exclude** - Will take a list to exclude results that match the conditions specified

Expression matching for these fields are as follows:

Pattern	Meaning
*	Matches everything
?	Matches any single character
[seq]	Matches any character in seq
[!seq]	Matches any character not in seq

Tip: You can use either commas and spaces in order to separate the patterns specified in these fields

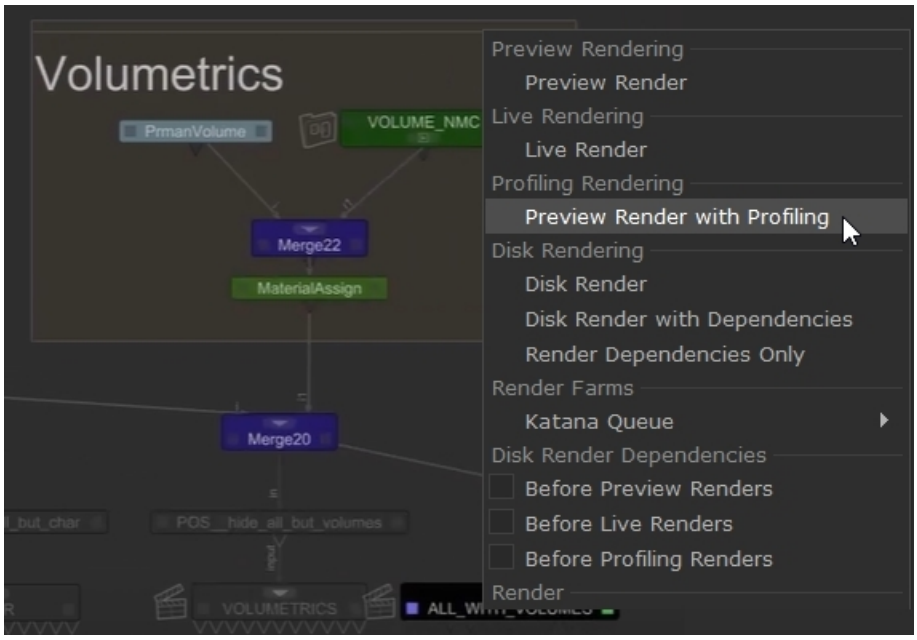
In the example below, we have set a filter for the _**NoName**_ entry, in order to delve into operations unrelated to nodes, such as implicit resolvers. See Implicit Resolvers for more information. For those wanting a breakdown of these _NoName_ operations, it is possible to set the **Show Results By** field to **Ops**, in order to view a breakdown of these implicit resolvers.



Optimizing Nodes with the Heat Map

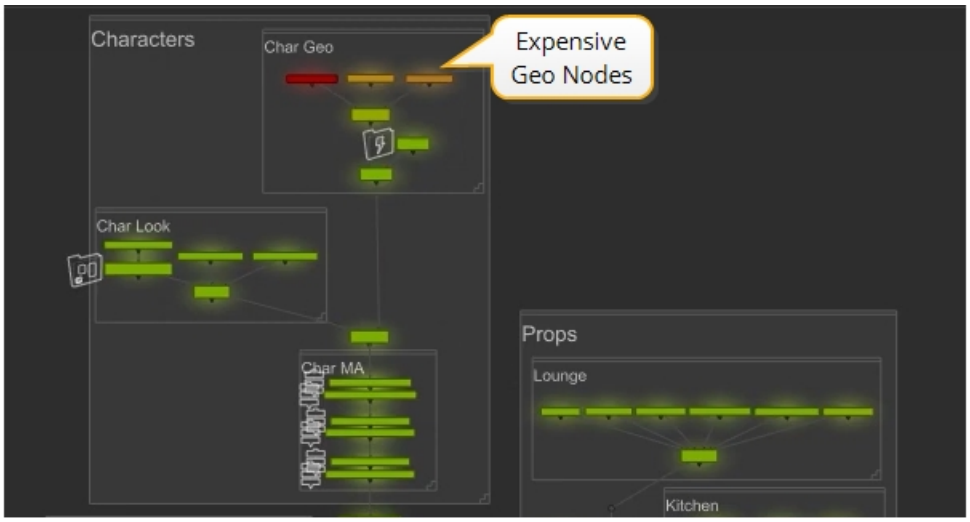
Katana's **Performance** tab heat map gives you a breakdown of your entire project at a glance. It's easy to locate CPU-heavy nodes so you can investigate where performance can be improved.

1. Render the scene by right-clicking a render node and selecting **Preview Render with Profiling**.



2. The **Performance** tab shows the time to first pixel rendered and a list of nodes and CPU usage, with the most expensive nodes at the top.
3. Enable **Show Heat Map** to display heat colors in the **Node Graph** tab.

   In this example, the **Char Geo** import nodes are clearly the most expensive nodes.

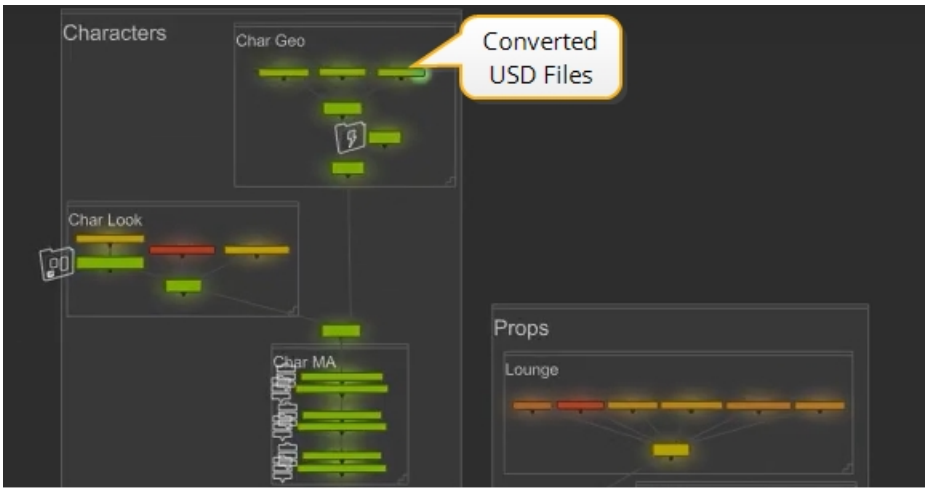


   Geometry nodes are often CPU-heavy, but in this case we've imported USD files that were encoded as **.usda**, which is not an optimal format for character animation.

   In this particular case, the way we have reduced CPU time is to convert the data into **.usd** files, which subsequently made the time against those geometry nodes more efficient. If we then recalculate the

Node Graph data in the **Performance** tab, we get a much more varied heat map. This is because the UsdIn nodes are no longer using way more resources that all other nodes, which evens out the heat map colors.

Note: Subsequent profile **.json** files don't overwrite each other, you get a new file for each calculation with a time stamp.



This is a very simple example, but it highlights the power of the heat map for locating bottlenecks in your Katana projects.

Phone a Friend if You Can't Troubleshoot Yourself

OK, so you've removed bottlenecks and reduced some of the CPU-heavy nodes' processing time, but your project is still not as optimal as it could be. Time to call for help. Katana saves performance data to Profile Results Files automatically, which you can share with colleagues if you're really stuck. Your colleague only needs your Katana project and your profile **.json** file to get started, they don't even have to perform a render because all the data they need is already in the **.json** file.

Note: If changes are made in the scene, the imported static profiling data becomes invalid. You'll need to run another **Preview Render with Profiling** to update the data to include any changes.

Profiling data is automatically saved as **.json** files to the KATANA_TMPDIR location, but you can specify a different directory at startup using the **--profiling-dir** command line argument. See Command-line Interface for more information.

1. Save the project and make a note of the Profile Results File location. Send the project and the **.json** file to your colleague.
2. On their machine, load the Katana project and open the **Performance** tab.
3. In the **Profile Results File** field, enter the location of the profiling **.json** or click the **Browse** dropdown to navigate to the location of the file.

   They don't need to execute the **Preview Render with Profiling** step themselves, the data is already available.
4. Enable **Show Heat Map**, if it isn't already checked.

   All the data is now available for your colleague to help you get back on track.

Using the Python Profiler

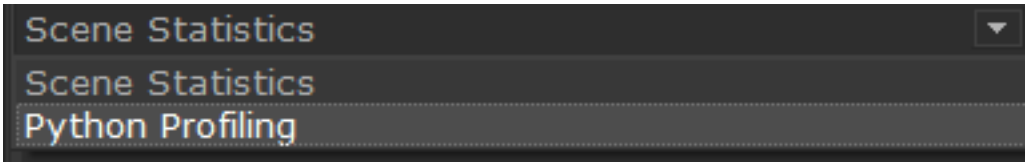
About the Python Profiler

The **Python Profiling** viewport provides a simple interface for collecting and displaying Python profiling statistics of your Katana workspace.

This is to further help you debug which part of Katana's user interface is causing slowdowns while an operation is being performed like changing a parameter for example.

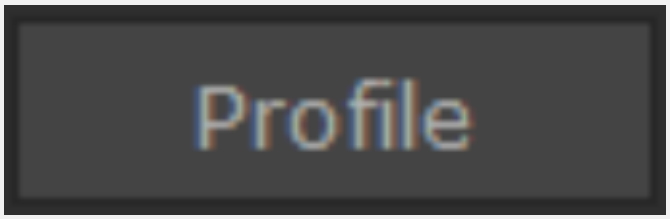
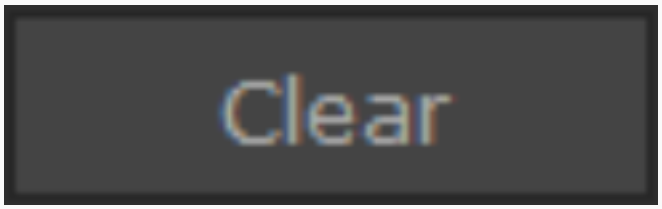
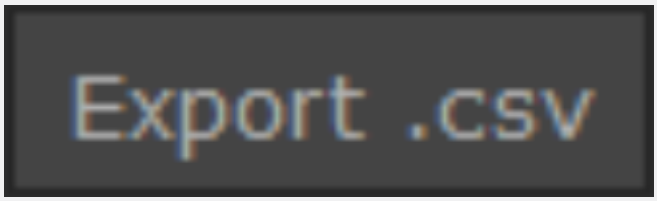
Navigating and Operating the Python Profiler

To open up the Python Profiling viewport, first navigate to the **Performance** tab and then select **Python Profiling** from the dropdown menu located at the top of the pane.



Tip: If a **Performance** tab does not exist in your Katana workspace, you can add it to your layout simply by navigating to the **Tabs** menu (**Tabs** > **Performance tab**) located in the menu bar at the top of your Katana workspace.

By default these are the standard controls displayed in the Python Profiling viewport.

Control	Function
	To start and stop a profile.
	Will clear the profile results listed in the table below once a profile has been completed.
	Opens a new window to export profile results as a spreadsheet at a specified directory location.

Below these controls are where the profile results will be listed in a table. These will only display once you have completed a profile.

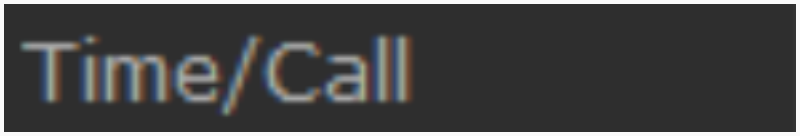
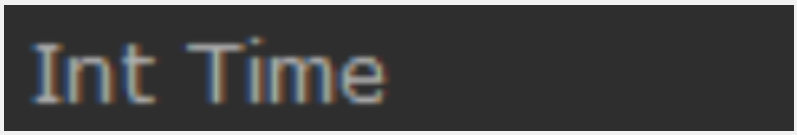
Each result will be displayed under each column and can be sorted by ascending or descending order. You can adjust this simply by clicking on the headers; indication arrows will be displayed on the right-hand side.



Indication arrow

Within the profile results table each column describes the following:

Header	Meaning
	Indicates which functions were called during a profile.

Header	Meaning
	Lists the number of calls made by each function.
	Displays the cumulative time made by each result.
	The quotient of cumulative time divided by the primitive calls (not listed).
	Internal time for each result.
	The quotient of internal time divided by the primitive calls (not listed).

Note: The headers and their meanings are based on the sortKeys as specified by the official Python Documentation.

Performing a Profile

1. To start a profile, click the **Profile** button and then proceed to perform some operations like changing a parameter.
2. To end a profile and display the results, click the **Profile** button again.

Note: To ensure the profile is running correctly, an indicator message located at the bottom left of the pane will change from **Click Profile to start profiling** to **Profiling...**

Viewing the Callers / Callees for each result

From the result table, you can access further details on each function.

Right-click on any result and select **Get callers...** or **Get callees....**

This will display a new window showing further information about the Callers and Callees contributing to the selected function call.



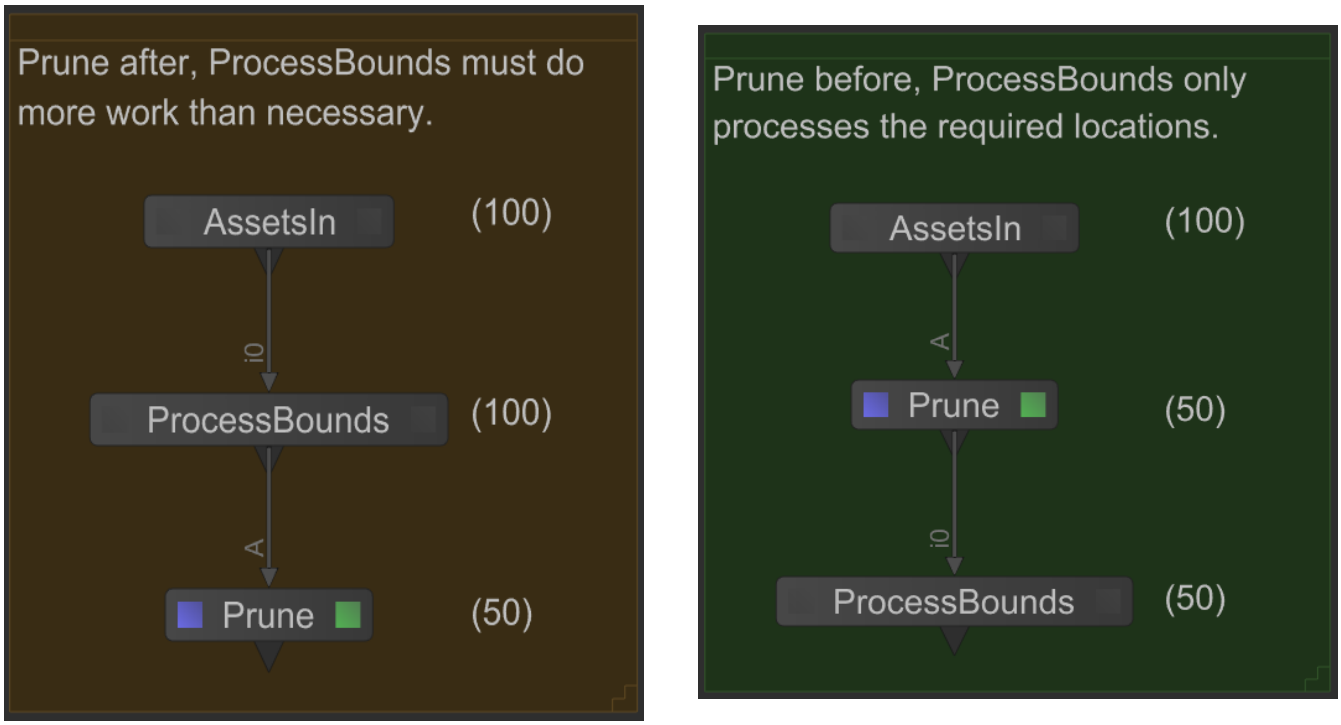
Note: Caller results are operations that the function has called, while **Callees** results are operations that called to the function prior.

Improving Your Node Graph

Prune Locations as Early as Possible

The **Prune** node is used to remove objects from the scene. Given the processing model used by Geolib3-MT, the sooner unneeded scene graph locations/objects are pruned from the scene, the less work required from both downstream Ops and Geolib3-MT itself. In this respect, Prune can be thought of as providing a filtering operation, reducing the size of the scene graph that must be processed by downstream Ops.

In the following example, assets are loaded by the **AssetsIn** node producing a scene graph of 100 locations. Bounding boxes are calculated by the **ProcessBounds** Op on all 100 locations and then unneeded scene graph locations for this shot are then pruned from the scene. However, if the Prune node is placed closer to the **AssetsIn** node, the scene graph processed by **ProcessBounds** is half the size. This directly translates to both memory and processing cost savings.



Prune scene graph objects/locations as early as possible. Doing so reduces the number of locations downstream Ops must process which corresponds to a reduction in both memory and scene processing time.

Understand Parallel Scene Processing Dimensions

Geolib3-MT's scene graph processing system searches for computationally independent tasks which can be evaluated in parallel. Broadly speaking, there are two dimensions of parallelism Geolib3-MT looks to exploit:

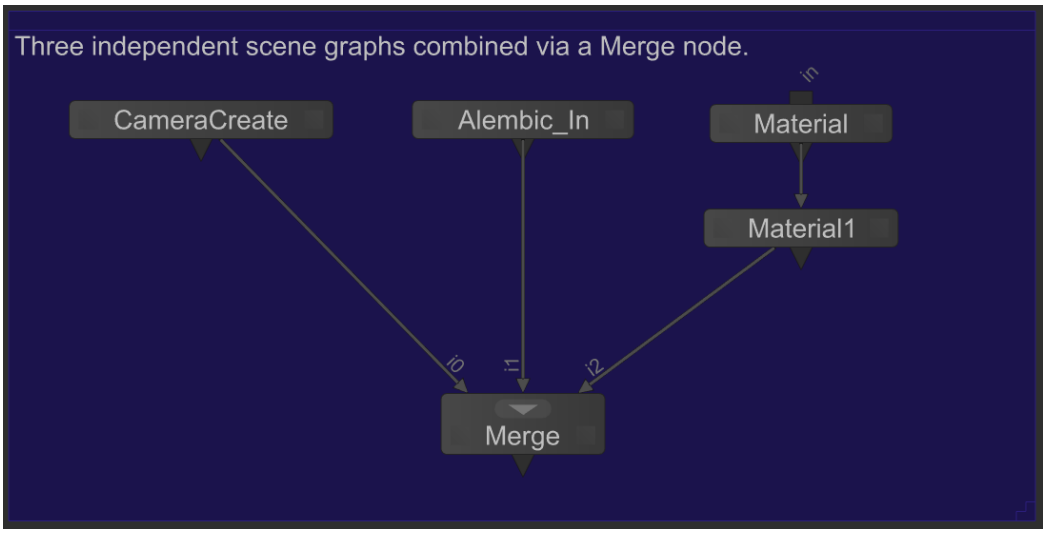
- **Scene Graph Parallelism** - In a deferred/lazily evaluated scene graph such as Katana, a scene graph contains potential work. It is potential because it is only computed once a scene graph location and its children become known through expansion.
  Each child of a scene graph location represents a computationally independent task, depending only on its parent location. Once a scene graph location has been computed, all of its children can be computed in parallel.

- **Op Tree Parallelism** - Sub sections of the Op tree can be processed in parallel. In doing so Geolib3-MT fully expands the scene at the sub section of the Op tree it's processing. Some ops are naturally parallelizable, such as source Ops (i.e. those Ops with no inputs). Some constructs are also naturally parallelizable, such as Merge Ops, in which each Op tree branch is independently computable.

Exploit parallelism in both dimensions to provide Geolib3-MT with the maximum backlog of known computationally independent tasks to process.

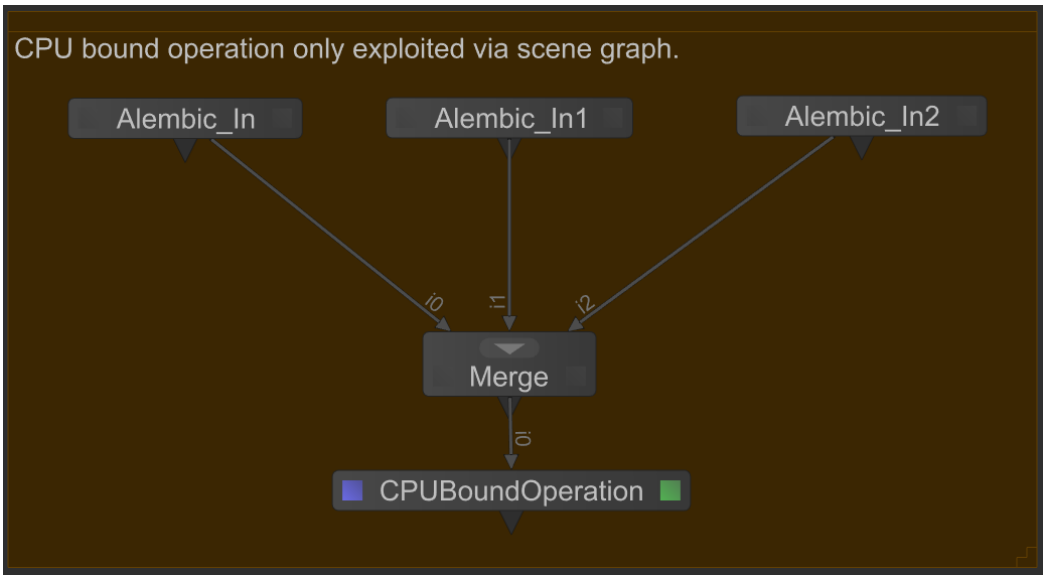
Use Merge Branches for Computationally Independent Scene Graphs

Each branch of a Katana node graph can be thought of as producing an independent scene graph, which are combined through the use of a Merge node.

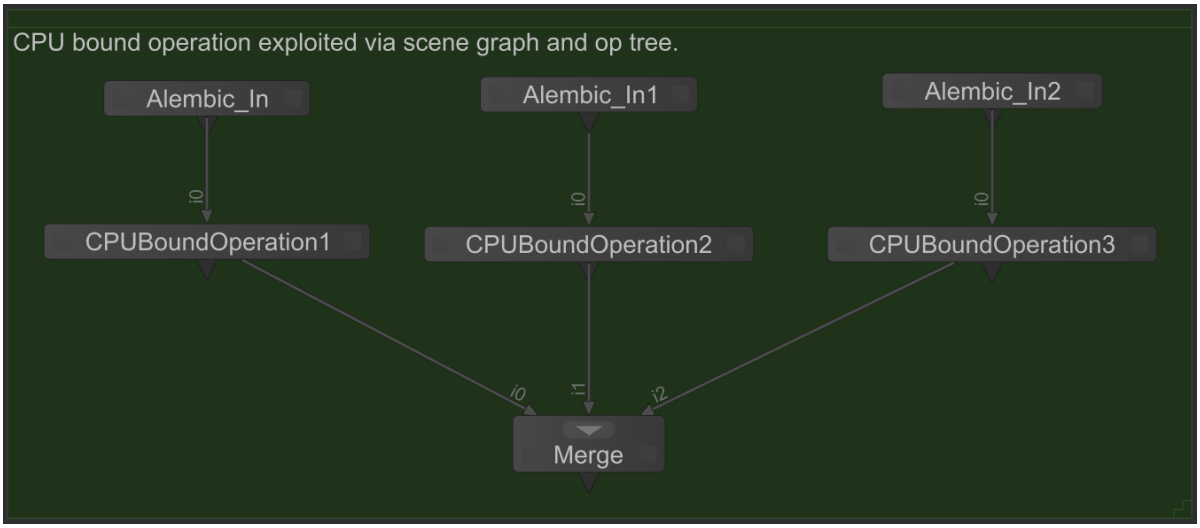


As each branch of the Merge node represents a computationally independent scene graph, Geolib3-MT exploits this fact to expand each branch in parallel. If this behavior is not desired, it can be disabled in the **RenderSettings** node. To benefit from this parallel expansion of Op tree branches there are a number of things to consider:

- Consider your scene graph not as one large scene graph but multiple smaller scene graphs, each produced by one or more connected Ops.
- Consider the data dependencies that exist in your scene graphs and the nodes an Ops responsible for producing them. Are there opportunities to refactor your node graph to take advantage of the parallel processing available from multiple independent branches?
- Identify CPU-bound operations that must operate on multiple locations. Whilst Geolib3-MT exploits parallelism available within the scene graph itself (by processing independent locations in parallel), CPU bound operations could also be evaluated in parallel by duplicating CPU bound nodes and ops.



In this example, whilst the three scene graphs produced by the **Alembic_In** nodes are generated in parallel, the parallelism available to the **CPUBoundOperation** node is limited to the scene graph only.



Example refactored node graph version in which the **CPUBoundOperation** is placed within each branch, which means that parallelism can be taken advantage of in both dimensions (scene graph and Op tree).

Place chains of collapsible Nodes/Ops together

In order to process a scene graph location at a particular Op, Geolib3-MT must perform a number of steps, including but not limited to:

- Ensuring the location's dependencies (parent location and the location it inherits its attributes from) have also been evaluated.
- Memory has been allocated to store the results of processing the scene graph location.

- Any inherited scene graph attributes have been applied prior to cooking the location.
- Evaluate the actual location using the Op's cook function.

Whilst this process is efficient, it is not cost-free. Checking dependencies requires checking a central cook result store, memory allocation requires a (potential) system call and the cook call incurs a small overhead to call the function (even if the **cook()** call doesn't mutate the scene graph).

With this in mind, anything that can be done to reduce the number of cooks has the potential to improve scene graph processing time and reduce memory usage.

Geolib3-MT's new Op tree optimization feature performs a preprocessing step to analyze the topology of the Op tree. One such optimization is the collapsing of chains of Ops of the same type. For example, a series of four **AttributeSet** Ops acting on the same set of locations may be collapsed into a single **AttributeSet** acting on that set of locations. This reduces the number of cook results in the caching subsystem, thus reducing the memory footprint of scene traversal.

Improving Your Ops

Consider using a scalable memory allocator

A memory allocator receives requests from the user application for memory (via **malloc()**) and responds with the address of a memory location that can be used by the application. When the application has finished with the memory it can return this memory to the allocator (via **free()**). Given the central role memory plays in most applications the performance of the memory allocator is of critical importance.

Most general purpose allocators that ship with operating system and the C Standard Library implementations can suffer from a number of pitfalls when used within highly multithreaded applications:

- More requests to the operating system for blocks of memory than required. Each request to the operating system results in a system call which requires the CPU on which the request is made to switch to a higher privilege level and execute code in kernel mode. If an allocator makes more requests to the operating system to allocate memory than is required the application's performance may suffer.
- Use of concurrency primitives for mutual exclusion. Many general purpose allocators weren't designed for use in multithreaded applications and to protect their critical sections, mutexes or other concurrency primitives are used to enforce correctness. When used in highly multithreaded applications the use of mutexes (either directly or indirectly) can negatively impact the application's performance.

The general purpose allocator, used on most GNU/Linux operating systems is ptmalloc2 or a variant of it. Whilst ptmalloc2 exhibits low memory overhead it struggles to scale as requests are made from multiple threads. Therefore, if profiling indicates your scene is spending a significant portion of time in calls to **malloc ()/free()** you may consider replacing the general purpose allocator with a scalable alternative.

Internally Geolib3-MT makes uses a number of techniques to handle memory allocation including the judicious of jemalloc to satisfy some requests for memory in the critical path.

Mark custom Ops as thread-safe where possible

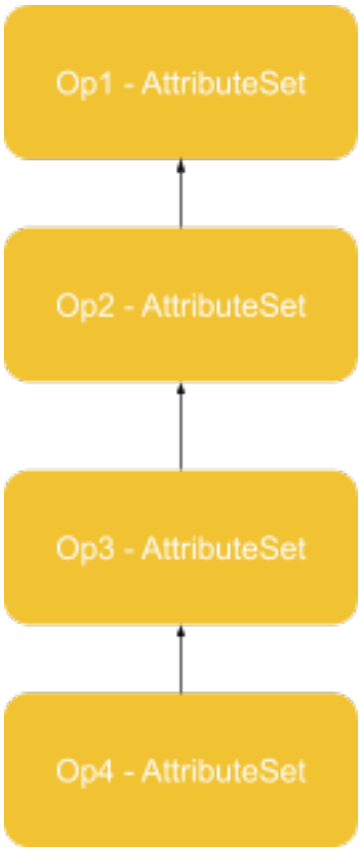
Geolib3-MT is designed to scale across multiple cores by cooking locations in parallel. An Op may declare that it is not thread-safe by calling **FnGeolibCookInterface::setThreading()**, in which case a Global Execution Lock (GEL) must be acquired while cooking a location with the Op. This prevents other thread-unsafe Ops from cooking locations, so is likely to cause pockets of inefficiency during scene traversal. In profiling your scenes, first identify and convert all thread unsafe Ops.

Thread-unsafe Ops can easily be identified from the **Render Log**: if a thread-unsafe Op is detected in the Op tree, a warning is issued:

```
[WARN plugins.Geolib3-MT.OpTree]: Op (<opName>: <opType>) is marked
     ThreadModeGlobalUnsafe - this might degrade performance.
```

Mark custom Ops as collapsible where possible

Real world scenes contain many instances of the following Op Chain construct:



This can arise for a number or reasons, given the versatility of the **AttributeSet** node, they are frequently used to **hotfix** scenes to ensure a given enabling attribute is present at render time. Or, the sequence of Ops can represent a set of logical steps to be taken during the creation of a given node graph. However, chains of similar Ops represent both a potential overhead and optimization opportunity for Geolib3-MT see Improving Your Ops .

Scenes may take advantage of Geolib3-MTs ability to optimize the topology of the Op tree. Custom Ops may indicate they can be collapsed by calling **FnGeolibCookInterface::setOpsCollapsible()**. This function takes an **FnAttribute::StringAttribute** as parameter, whose value indicates the name of the attribute to which the collapsed Ops' arguments are passed.

For example, consider the chain of four **AttributeSet** ops above. Geolib3-MT collapses this chain of Ops by gathering the Op args for each Op into a **GroupAttribute** called **batch**, whose children contain the Op args of each op in the chain.

- batch
  - Op1
  - Op2
  - Op3
  - Op4



Note: The Op args are passed to the collapsed Op in top down order.

Custom user Ops can participate in the Op Chain collapsing functionality by calling **setOpsCollapsible()** as described above. In the above example **AttributeSet** calls **setOpsCollapsible("batch")** and then tests whether it's running in batch mode during its **cook()** call.

If participating in the Op Chain collapsing system, the Op makes the firm guarantee that the result of processing a collapsed chain of Ops must be identical to processing each one in sequence. The implications of this mainly relate to the querying of upstream locations or attributes where those locations or attributes could have been produced or modified by the chain that has been collapsed. As a concrete example, consider a two-Op Chain in which the first Op sets an attribute **`hello=world`** and the second Op in the chain prints **`Hi There!`** if **`hello`** is equal to **`world`**. If the second Op uses **Interface::getAttr()** to query the value of **`hello`** in the collapsed chain the result is empty, as hello has been set on the location's output but did not exist on the input. To remedy this, the op can be refactored to call **getOutputAttr()** instead.

Cache frequently accessed attributes

While access of an **FnAttribute**'s data is inexpensive, retention or release of an **FnAttribute** objects requires modifying a reference count. This is not typically a problem however, this may accumulate for Ops that create many temporary or short lived **FnAttribute** objects, especially if many threads are executing the Op at once. In cases where an **FnAttribute** instance is frequently used, consider whether it may be cached to avoid this overhead.

As a concrete example, consider the following OpScript snippet:

```
local function generateChildren(count)
      for i=1,count do
            local name = Interface.GetAttr("data.name").getValue() .. tostring
(i)
            Interface.CreateChild(name)
      end
end
```

This function creates count children, with names derived from the input attribute **data.name**. If count is large, accessing this attribute inside the loop causes unnecessary work for two reasons,

- Lua must allocate and deallocate an object for the **data.name** attribute each iteration, causing more work for the Garbage Collector.
- This allocation and destruction causes an increment and decrement in the attribute's atomic reference count, potentially introducing stalls between threads.

The second bottleneck applies when OpScripts accessing the same set of attributes are executed on many threads. To avoid accessing the reference count so frequently, this snippet may be rewritten as follows:

```
local function generateChildren(count)
      local stem = Interface.GetAttr("data.name").getValue()
      for i=1,count do
            local name = stem .. tostring(i)
            Interface.CreateChild(name)
      end
end
```

Now **data.name** is accessed only once, outside of the loop, so the reference count is not touched while the loop executes.

Composing Concurrency-Friendly Scenes

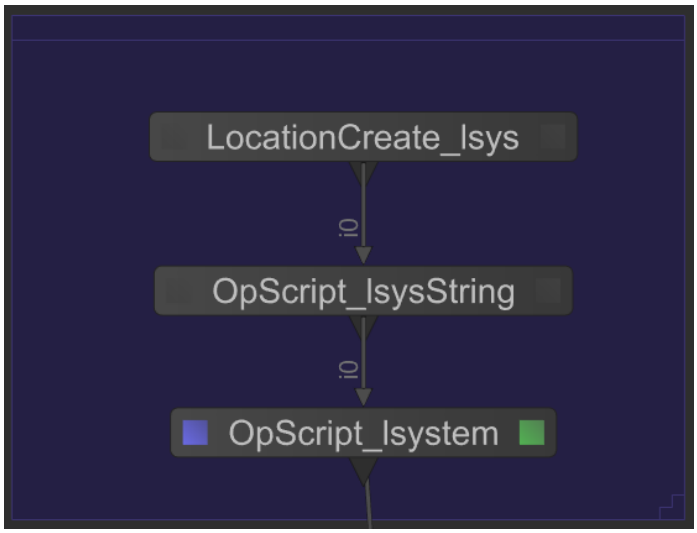
Understand Scene Throughput and How to Identify Bottlenecks

In Geolib3-MT, system throughput is defined as the number of scene graph locations that are processed per second. In processing a given scene graph the moment-to-moment objective is to either maintain or increase the Geolib3-MT processing throughput.

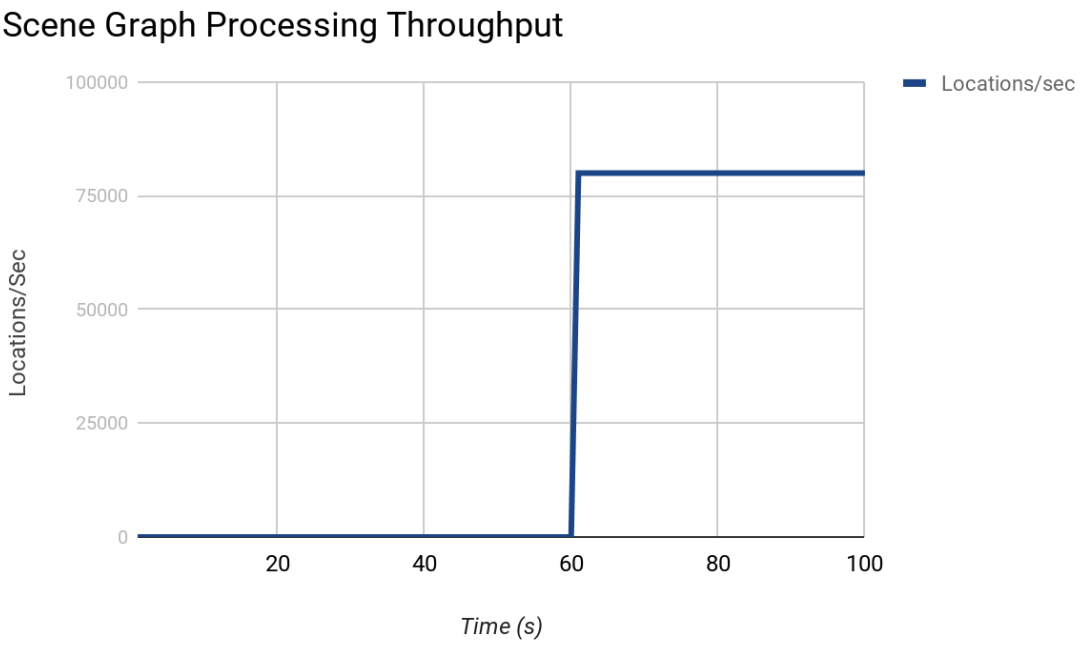
Location Bottlenecks

In a typical scene, some scene graph locations take longer to compute than others, this is normal and expected behavior of scene graph processing. However, these expensive-to-compute locations can become system-wide bottlenecks when subsequent locations depend on their results.

To provide a concrete example, in the following screenshot OpScript_lsysString produces a costly L-system description at the scene graph location /root/world/geo/lsys. The node OpScript_lsystem then processes this description to build a graphical representation of the tree.



Because the work of OpScript_lsystem is entirely dependent upon OpScript_lsysString, processing of the whole scene graph is stalled until /root/world/geo/lsys is computed by OpScript_lsysString. The resulting throughput when processing this tree is approximate to:



On initial inspection it could be reasonable to assume the processing between 0-60s is bounded by a mutex or serial code. But, as previously discussed, this is actually a very costly location upon which the rest of the scene depends on. What options exist to address scene bottlenecks like this?

First, we can use the Op tree profiling tools shipped with Katana 3.5 to identify costly Ops in the scene. Using this information we can seek to optimize the performance of those costly Ops. Optimizing these Ops largely

depends upon the nature of the work they're doing; but as with any optimization task, first identify an optimization target, measure, refactor and re-measure.

It may be possible to entirely remove the bottleneck by just optimizing one or two key Ops in the Op tree. This was the case in the above example in which careful use of Lua object creation significantly reduced the time to compute the L-system description. However, sometimes we must accept the operation simply is costly; in this case, what options exist to help maintain scene processing throughput?

The next step is to examine the processing problem itself, can it be broken down into smaller, logical chunks that could be linked or indexed by scene graph locations. Effectively using the scene graph topology to represent a parallel processing task graph where each child represents a subsection of the work that can be processed in parallel.

For example, say we need to compute the first 1000 prime numbers. We could create an Op that at the location /root/world/geo/data/primes calculates each prime in turn and stores them as an IntAttribute. Alternatively, we could create an Op that creates 1000 children under /root/world/geo/data/primes, each child's position serving as the index to the prime series. At those children, the Op would then compute the prime at that point in the series. In this example, Geolib3-MT is able to take advantage of the parallelism available within the scene graph to schedule each child to run in parallel.

An extension to this approach is also provided in Use Merge branches for computationally independent scene graphs in which computationally intensive work is broken up using the Op tree and the results collated using a Merge node.

Finally, you can use knowledge of the scene traversal pattern to your advantage. Geolib3-MT completes a breadth first expansion of the scene, thus, if there exists a computationally expensive scene graph location, consider placing it under a scene graph location with a number of other children which can also be processed in parallel. Then, whilst one location may take longer to cook, it does not prevent the other locations—indeed, the rest of the scene graph—from being evaluated.

Processing Capacity

Scene throughput can be bounded by processing capacity. If there are more scene graph locations to be processed than available CPU cores, we would expect to see complete utilization (saturation) of available cores and a backlog of scene graph locations awaiting processing. In general, adding more cores should help reduce scene graph processing time.

Conversely, if the available computing resources are able to process scene graph locations faster than new locations become known, or, if the scene simply does not contain a large enough ("large enough" also depending on what processing Ops in the Op tree are doing at each scene graph location) number of scene graph locations to keep all cores utilized you may benefit from reducing the number of cores available to fully utilize those cores. From a performance standpoint this may be beneficial when combined with a CPU pinning strategy which could potentially improve the L2/L3 cache performance.

Identify Indirect Use of Mutexes and Other Synchronization Primitives

Mutexes and other synchronization primitives that enforce serialized execution of code can have a significant and negative impact on scene graph evaluation performance. Whilst it may be straightforward to identify mutexes in your own code it can be more problematic in third-party code and libraries.

Symptoms that Ops in your scene may be indirectly calling code wrapped in mutexes or critical sections range from:

- Increasing the number of threads/cores available for scene processing increases, rather than decreases, scene graph expansion and time to first pixel.
- Messages in the Render Log that indicate a thread-unsafe Op has been called.
- Multiple threads/cores available for scene processing but only one or two cores actually utilized.

Alembic HDF5 to Ogawa

To provide a concrete example, Alembic caches can be stored in two formats, HDF5 and the newer Ogawa format which was designed to support parallel scene traversal. Whilst no messages are printed to the **Render Log** when reading HDF5 formatted caches, the scene traversal time is significantly slower and performance may even degrade when more threads are added. In a test scene with approximately 140k locations, converting an existing HDF5 cache to Ogawa saw a reduction in scene traversal time from 26.2s to 3.9s with a 32 core task pool available for processing.

The Alembic library provides a utility to convert Alembic files from HDF5 to Ogawa, called abcconvert. This utility must be built from source; this source can be found at here, with instructions in the root of the repository.

Once built, abcconvert can be used from the command line as follows:

- cd /path/to/build/directory
- abcconvert -toOgawa -in <input file> <output file>

abcconvert does nothing if the input file is already in Ogawa format.

Improving OpScript Performance

Understand the Lua Garbage Collector

In Lua, memory can be allocated on either the stack or the heap. Objects or memory allocated on the heap is reclaimed through Lua's garbage collector. When the garbage collector is running your Lua code cannot make forward progress therefore, reducing the frequency with which the garbage collector must run or, the amount of garbage it must collect, helps to improve the performance of your OpScripts.

The following Lua constructs, while not bad, result in a new object being created which must be subsequently cleaned up during garbage collection. You should pay particular attention it these constructs are used within loops.

- **String_one..string_two** - String concatenation or any string creation function could potentially result in the creation of a new object. As Lua strings are unique it firstly checks to determine if the string has been created elsewhere.
- **{ ... }** - Each time a table constructor is executed, a new table is created.
- **function() ... end** - Executing a function statement creates a closure. If this executed within a loop of n iterations it results in the creation of n closures.



Note: For more information see lua-users.org.

Avoid ".." in loops

As a concrete example of Understand the Lua Garbage Collector, in this recommendation we explore the use of the string concatenation operator "..", commonly used in OpScripts due to its convenient shorthand notation.

As an example, consider the following code used to build an L-System description:

```
local result = ""
for i = 1, #inputStr do
    local char = inputStr:sub(i,i)
    local ruleStr = rulesTable[char]
```

```
      if ruleStr then
            result = result..ruleStr
      else
            result = result..char
      end
end
return result
```

`result` is appended to in a tight loop. Whilst convenient, this results in a large number of string allocations and poor performance. The preceding code can be re-written to handle all concatenation only once the loop has completed as shown below:

```
local buf = {}
for i = 1, #inputStr do
      local char = inputStr:sub(i,i)
      local ruleStr = rulesTable[char]
      if ruleStr then
            buf[#buf+1] = ruleStr
      else
             buf[#buf+1] = char
      end
end
return table.concat(buf)
```

Running the above example as part of a large Katana scene file reduced the scene processing time of this function by approximately 2.5x as shown in the following table:

String Method	Time
".." Operator	7.011s
Table.concat()	2.681s

Consider creating custom C++ Ops to replace costly OpScripts

While Lua and the OpScript API are very convenient and versatile, OpScripts that account for a large portion of scene traversal time may be better suited as custom C++ Ops.

As a concrete example, consider the following OpScript, which places the target location at a random position, orientation and scale within a uniform box.

```
Local hash = ExpressionMath.stablehash(Interface.GetOutputName())
math.randomseed(hash)

local dim = 10
local s = 0.15

local sx = s*(1 + 0.5*math.random())
local sy = s*(1 + 0.5*math.random())
local sz = s*(1 + 0.5*math.random())

local tx = 2*dim*(math.random() - 0.5)
local ty = dim* math.random() + sy
local tz = 2*dim*(math.random() - 0.5)

local ax = math.random()
local ay = math.random()
local az = math.random()
local axis = Imath.V3d(ax, ay, az):normalize()
local angle = 360 * math.random()

local translate = Imath.V3d(tx, ty, tz)
local rotate = Imath.V4d(angle, axis.x, axis.y, axis.z)
local scale = Imath.V3d(sx, sy, sz)

Interface.SetAttr("xform.group0.translate",
                  DoubleAttribute(translate:toTable(), 3))
Interface.SetAttr("xform.group0.rotate",
```

```
                DoubleAttribute(rotate:toTable(), 4))
Interface.SetAttr("xform.group0.scale",
                DoubleAttribute(scale:toTable(), 3))
```

(Such an OpScript is of course contrived, but one could imagine replacing randomness with a well-defined distribution to place geometry.)

This OpScript could be rewritten as the following C++ Op:

```
#include <FnAttribute/FnAttribute.h>
#include <FnGeolib/op/FnGeolibOp.h>
#include <FnGeolibServices/FnExpressionMath.h>
#include <FnGeolibServices/FnGeolibCookInterfaceUtilsService.h>
#include <FnPluginSystem/FnPlugin.h>

#include <cstdlib>

inline double getRandom()
{
     return (double)rand() / (double)RAND_MAX;
}
struct DistributeGeometryOp : public Foundry::Katana::GeolibOp
{
     static void setup(Foundry::Katana::GeolibSetupInterface &interface)
     {
          interface.setThreading
(Foundry::Katana::GeolibSetupInterface::ThreadModeConcurrent);
     }
     static void cook(Foundry::Katana::GeolibCookInterface &interface)
{
srand(FnGeolibServices::FnExpressionMath::stablehash(interface.getOutputName
()));
          double dim = 10.0;
          double s = 0.15f;
          double sx = s*(1.0 + 0.5f*getRandom());
          double sy = s*(1.0 + 0.5f*getRandom());
          double sz = s*(1.0 + 0.5f*getRandom());
```

```
            double tx = 2.0f*dim*(getRandom() - 0.5f);
            double ty = dim* getRandom() + sy;
            double tz = 2.0f*dim*(getRandom() - 0.5f);

            double ax = getRandom();
            double ay = getRandom();
            double az = getRandom();

            double axisLen = sqrt(ax*ax + ay*ay + az*az);
            ax /= axisLen;
            ay /= axisLen;
            az /= axisLen;

            double angle = 360.0 * getRandom();

            double translate[] = { tx, ty, tz };
            double rotate[] = { angle, ax, ay, az };
            double scale[] = { sx, sy, sz };
            interface.setAttr(
                  "Xform.group0.translate",
                  FnAttribute::DoubleAttribute(translate, 3, 3));
            interface.setAttr(
                  "Xform.group0.rotate",
                  FnAttribute::DoubleAttribute(rotate, 4, 4));
            interface.setAttr(
                  "Xform.group0.scale",
                  FnAttribute::DoubleAttribute(scale, 3, 3));
            }
};
DEFINE_GEOLIBOP_PLUGIN(DistributeGeometryOp);
void registerPlugins()
{
      REGISTER_PLUGIN(DistributeGeometryOp, "DistributeGeometryOp", 0, 1);
}
```

Up to a different random distribution, these Ops produce the same scene. However, the C++ version executes around 3.2 times as fast: when used to position 10,000 geometric primitives, the Ops use the following resources:

Op	CPU time	Memory used
OpScript	0.989s	54.24 MiB
C++ Op	0.310s	53.67 MiB

In this instance, memory usage is similar. However, if the OpScript is allocating and freeing a lot of small objects, converting to C++ also saves time spent in Lua garbage collection.

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		
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.
numberOfRecentProjects	10	The number of recent projects that are shown in the **File** > **Open Recent** menu. The maximum number of recent projects that are stored is 100.
crashFile		
numberOfActions	50	Set the number of actions before automatically saving the current project to a file from which

Control (UI)	Default Value	Function
		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.
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.

Control (UI)	Default Value	Function
		• **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.
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.

Attributes

Control (UI)	Default Value	Function
attributes		
showInternalNodesInAttributeHistory	No	Show internal nodes in Attribute History in the **Attributes** tab.

Color

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

Documentation

Control (UI)	Default Value	Function
documentation		
source	Local	Controls where Katana's documentation should be served from. • **remote** - uses help.thefoundry.co.uk/katana. • **local** - uses a local copy of the documentation.

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.

Control (UI)	Default Value	Function
flipbook		
colorspace	n/a	Default output colorspace for flipbook images.
format	n/a	Default file format for flipbook images.

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

Control (UI)	Default Value	Function
		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.
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 Common Parameter Widgets.
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		
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.
defaultShadingNodeViewState	Expanded	Specify whether new nodes added to a shading network are expanded or

Control (UI)	Default Value	Function
		collapsed by default. When **showPagesConnectedOnly** is enabled, setting this control to **Connected Only** displays the whole page containing connected inputs and outputs not just the connected inputs and outputs themselves.
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.
showOffscreenFlagArrows	Yes	Specify whether or not to display off-screen flag arrows.

Control (UI)	Default Value	Function
showPagesConnectedOnly	No	Specify whether the whole page or just the connected inputs and outputs are displayed when **defaultShadingNodeViewState** is set to **Connected Only**.
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).
useColorFromInputPortForConnections	Yes	Specify whether the color of a connection is determined by the color of the input port on the target node or the output port on the source node.

Nodes

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

Parameters

Control (UI)	Default Value	Function
parameters		
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).
stickyScrub	No	Allow modifying a number parameter by clicking once on its label and moving the mouse. A second click commits the modification.

Python

Control (UI)	Default Value	Function
python		

Control (UI)	Default Value	Function
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.
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		
findLocationsFilterDefault	Selected	Sets the default filter used by the **Find Scene Graph Locations** pop-up in the **Scene Graph** tab.
showLightsColumn	Yes	Choose whether or not to display the **Lights** column in the **Scene Graph** tab.
showRenderColumn	Yes	Determine whether to show **Render** Working Set column in the **Scene Graph** tab. The title of the **Render** column can be clicked

Control (UI)	Default Value	Function
		to turn the use of the render Working Set on. The render Working Set determines the scene graph locations for which geometry is rendered in Preview Renders and Live Renders. The **Monitor** tab contains a corresponding button to turn the use of the render Working Set on, in order to render only objects from the **Render** column of the **Scene Graph** tab. The button is only shown when this preference is set to **Yes**.
showViewerVisibilityColumn	Yes	Show **Viewer Visibility** Working Set column in the **Scene Graph** tab. The title of the **Viewer Visibility** column can be clicked to turn the use of the viewerVisibility Working Set on. The viewerVisibility Working Set determines the scene graph locations for which geometry is drawn in the **Viewer** tab. The **Viewer** tab contains a corresponding button to turn the use of the viewerVisibility Working Set on, in order to show only objects from the **Viewer Visibility** column of the **Scene Graph** tab. The button is only shown when this preference is set to **Yes**.
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		
antiAliasing	Full	Defines the number of samples used for multi-sample anti-aliasing. When set to Full, a maximum of 16 samples will be used, depending on the capability of your workstation's graphics card. When set to Medium or Low, 1/2 or 1/4 of the number of samples that would be used when set to Full will be used, so up to 8 and 4 samples, respectively.
continuousManipulation	Yes	Enables continuous manipulation of manipulators beyond the Viewer boundaries. Allows the mouse pointer to wrap around the viewport boundaries so that mouse move events are not limited by the screen dimensions. This option may not be suitable for remote desktop sessions where both Katana and the remote desktop client may compete for programmable movement of the mouse pointer to arbitrary positions. In such case, disabling this option may be desirable.
manipulationFreezePeriod	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.
viewer > **backgroundColor**	0.0400 0.0400 0.0400 1.0000	Color of the background.
viewer > **faceSelectModeColor**	0.0000 0.1300 0.2200 1.0000	Color of objects in face select mode.

Control (UI)	Default Value	Function
viewer > **faceSelectionColor**	0.7900 0.2200 0.0300 1.0000	Color of the selected faces.
viewer > **manipulators**		
scale	1	Global scale for the manipulators' size.
viewer > **monitorLayer**		
highlightColor	0.7157 0.4452 0.1356 1.0000	Color of the highlight on hover location during ID Buffer mode.
selectionColor	0.5906 0.3140 0.0742 1.0000	Color of the selection during ID Buffer mode.
viewer > **screenWindow**		
dashed	Yes	Specified whether the screen window should be drawn as a dashed line, rather than a solid line.
lineWidth	2	The width of the line to use to draw the screen window.
color	0.0000 0.3300 0.0900 0.5000	Color to use to draw the screen window.
viewer > **selectionColor**	1.0000 1.0000 1.0000 1.0000	Color of the selected locations.
viewer > **snapping**		
hitAreaSize	100	All geometries that fall within this area will be considered for snapping.
orientAxis	+Z	Axis and direction along which the object will be oriented
upAxis	+Y	Up direction for normal orientation
hitAreaColor	0.8710 0.7100 0.4040 1.0000	Color of the Snapping hit area

Control (UI)	Default Value	Function
wireframeColor	0.3000 0.3000 0.3000 1.0000	Color of the wireframe of the targeted object.

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.



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.

General Shortcuts

Keyboard Shortcut(s)	Action
Ctrl+****	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**).

Keyboard Shortcut(s)	Action
Ctrl+**Z**	Undo the last action.
Esc	Cancel the current render.
Shift + **Esc**	Cancel all renders.
F4	Show shelf items.
F5	Flush caches.
F6	Toggle implicit resolvers in the **Scene Graph** tab.
F7	Toggle the **Render Only Selected Objects** option in the **Scene Graph** tab.
F8	Toggle the **Auto-Key All** option.
Ctrl + **P**	Start a **Preview** render from the current view node.
Ctrl + **Shift** + **P**	Start a **Live** 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 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 Shortcuts

Keyboard Shortcut(s)	Action
A	Frames all selected.
D	Shows domain slider.
Delete	Deletes keyframe.
Esc	Exits insert mode.
F	Frames selected.
H	Shows heads up labels.
Insert	Enters insert mode.
S	Cycle snapping mode.

Dope Sheet 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 Shortcuts

Keyboard Shortcut(s)	Action
Ctrl+**A**	Select all.
Crtl+**C**	Copy message.
Delete	Delete message.

Monitor Shortcuts

Keyboard Shortcut(s)	Action
Period (**.**)	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+Up Arrow/Down Arrow	Increment/decrement display exposure (fstop) offset by 1/4 stop.

Keyboard Shortcut(s)	Action
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.
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).
S	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.
Shift+**Page Up**	Selects the previous available AOV.
Shift+**Page Down**	Selects the next available AOV.
Shift+**Home**	Toggles between the default and last selected AOV.

Node Graph Shortcuts

Keyboard Shortcut(s)	Action
Period (**.**)	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+**.** (period)	Dims the nodes that are unconnected to the viewed node.
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).

Keyboard Shortcut(s)	Action
Ctrl+MMB **Ctrl**+Return	Enter group under mouse pointer.
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**.
Ctrl+**Delete**	Holding **Ctrl** while deleting a node stops any upstream or downstream nodes from being reconnected through the deleted node.
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**.
F2	When a single node is selected, opens a pop-up to edit the name of that node.
G	Creates a **Group** node with the currently selected nodes moved inside.
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**.
L	Rearranges selected nodes, in the **Node Graph**, for clarity.
M	Merges the selected nodes by connecting their outputs to a Merge node. This functionality is the same as selecting **Edit** > **Merge Selected Nodes** from the **Node Graph** tab menu.
N	Displays the right-click node creation menu at the current mouse location.
P	Creates PrimitiveCreate nodes in the Node Graph, with their name and

Keyboard Shortcut(s)	Action
	nodeType parameters set to match a specific type of primitive.
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.
Return or **Enter**	When the pointer is over a node, opens a pop-up to edit the name of that node.
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.

Shading Node Shortcuts

Keyboard Shortcut(s)	Action
Alt+**1**	Fully collapse selected shading nodes' parameters and connections.
Alt+**2**	Expand selected shading nodes' connected ports only.

Keyboard Shortcut(s)	Action
Alt+**3**	Fully expand selected shading nodes' parameters and connections.
Alt+**Enter**	Show or hide the **Filter** function on selected shading nodes.
Alt+**H**	Show or hide the connections between selected shading nodes. **Tip:** Hold **Alt**+**H** with no selection to temporarily show hidden connections.
X	Pressing and holding the **X** key on a shading node will temporarily expand it. It will collapse when released.

Parameters Shortcuts

Keyboard Shortcut(s)	Action
D	Toggles the disabled state of node.
Ctrl+**F**	Opens a **Search** dialog for the tab.
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 Shortcuts

Keyboard Shortcut(s)	Action
Alt+Up Arrow	Previous history.
Alt+Down Arrow	Next history.
Ctrl+**X**	Cut.

Keyboard Shortcut(s)	Action
Ctrl+**C**	Copy.
Ctrl+**V**	Paste.
Ctrl+Return	Execute entered script.

Scene Graph Shortcuts

Keyboard Shortcut(s)	Action
Ctrl + **+**	Expands All/Expands branch.
Ctrl + **-**	Collapses All/Collapses branch.
+	Expands location(s).
-	Collapses location(s).
Left Arrow	Selects parent location of focused location/Collapses location.
Right Arrow	Selects first child location of focused location/Expands location.
Ctrl+Left Arrow	Selects parent locations of selected child locations.
Ctrl+Right Arrow	Selects all child locations of selected parent locations.
Ctrl+Up Arrow	Moves selection up.
Ctrl+Down Arrow	Moves selection down.
Click	Working Sets: Sets states of selected locations to be **Included**.
Shift+Click	Working Sets: Sets states of selected locations to be **Included with Children**.
Ctrl+Click	Working Sets: Sets states of selected locations to be **Excluded**.
Ctrl+**Shift**+Click	Working Sets: Sets states of selected locations to be **Excluded with Children**.
Alt+Click	Working Sets: Resets states of selected locations to their minimum allowed states.

Timebar 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.
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 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.

Keyboard Shortcut(s)	Action
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.
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.
V	Activate Snapping mode in the Hydra Viewer.
L	Activate Lighting Tools in the Hydra Viewer.
I	Activate Image-Based Selection in the Hydra Viewer.
`	Turn on and off the Monitor Layer in the Hydra Viewer.
Shift+**Page Up**	When the Monitor Layer in the Hydra Viewer is enabled  , this selects the

Keyboard Shortcut(s)	Action
	previous available AOV.
Shift+**Page Down**	When the Monitor Layer in the Hydra Viewer is enabled  , this selects the next available AOV.
Shift+**Home**	When the Monitor Layer in the Hydra Viewer is enabled  , this toggles between the default and last selected AOV.
C	Toggles between the perspective and orthographic views for the default **persp** camera.
D	Makes rotate (world) manipulators appear. Pressing **D** a second time makes the rotate (world) manipulators appear around the COI.
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. Pressing it repeatedly toggles between the Translate (world) and Translate Around COI manipulators.
Shift+**~**	Displays annotations.
T	Changes background color to black.
Alt+**T**	Changes background color to gray.

Keyboard Shortcut(s)	Action
Shift+**T**	Changes background color to white.
Tab	Displays the manipulator measurement tool.
U	Unhides selection.
V	Look through the selected object.
W	Makes transform manipulators appear.
X, **Y**, or **Z**	Switches the camera to look along the positive axis, aligned with the Center of Interest. The COI position and distance are maintained and the camera can be panned and tumbled as a perspective camera.
Shift+**X**, **Y**, or **Z**	Switches the camera to look along that negative axis, aligned with the Center of Interest. The COI position and distance are maintained and the camera can be panned and tumbled as a perspective camera.

Reference Guide

This manual aims to provide a complete reference for all the controls within each node in Katana. It does not give you any instructions on using Katana. For details on installing and using Katana, read the User Guide.

2D Nodes

The nodes in this section are 2D nodes that you can use within Katana. These are listed alphabetically, and each node includes a short description followed by a list of the node's parameters and their functions.

Color Nodes

The following section describes Katana's 2D **Color** nodes.

ImageBackgroundColor

Controls the background color of the input image using RGB, HSL, and/or HSV parameters.

Connection Type	Connection Name	Function
Input	input	The image sequence the colors of which you want to modify.

Control (UI)	Default Value	Function
color		
color	0.0, 0.0, 0.0, 1.0	The color (RGBA values) of every pixel in the image. You can also use the below RGB, HSL, or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.

ImageBrightness

This node multiplies the image's RGB channels to increase or decrease brightness.

Connection Type	Connection Name	Function
Input	input	The image sequence whose brightness you want to adjust.
	out_mask	An optional image to use as a mask. By default, the brightness change is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageBrightness parameters continued		
brightness	1	Adjusts the brightness of the rgb channels in the image.
alpha	1	Adjusts the brightness of the alpha channel in the image.

ImageChannels

This node lets you:

- rearrange up to 4 channels from a single image (one input)
- combine channels from several inputs into one output. For example, you can use it to combine two separate passes (such as the beauty pass and the reflection pass) into the same data stream.
- replace a channel with luminance, black (removing the alpha channel, for example), white (making the alpha solid, for example), or any other constant color.

Connection Type	Connection Name	Function
Input	i0 You can add as many numbered input ports as you want by pressing ▼ in the node.	The image sequence the channels of which you want to modify.

Control (UI)	Default Value	Function
redSource	i0	Select the input from which to take the red channel.
redChannel	R	Select what to use as the red channel: • **R** - use the red channel from **redSource**. • **G** - use the green channel from **redSource**. • **B** - use the blue channel from **redSource**. • **A** - use the alpha channel from **redSource**. • **Lum** - use the luminance from **redSource**. • **1** - set the red channel to white. • **0** - set the red channel to black. • **Const** - set the red channel to any constant color. You can select the color using the **constantColor** controls.
greenSource	i0	Select the input from which to take the green channel.
greenChannel	G	Select what to use as the green channel: • **R** - use the red channel from **greenSource**. • **G** - use the green channel from **greenSource**. • **B** - use the blue channel from **greenSource**. • **A** - use the alpha channel from **greenSource**. • **Lum** - use the luminance from **greenSource**. • **1** - set the green channel to white. • **0** - set the green channel to black.

Control (UI)	Default Value	Function
		• **Const** - set the green channel to any constant color. You can select the color using the **constantColor** controls.
blueSource	i0	Select the input from which to take the blue channel.
blueChannel	B	Select what to use as the blue channel: • **R** - use the red channel from **blueSource**. • **G** - use the green channel from **blueSource**. • **B** - use the blue channel from **blueSource**. • **A** - use the alpha channel from **blueSource**. • **Lum** - use the luminance from **blueSource**. • **1** - set the blue channel to white. • **0** - set the blue channel to black. • **Const** - set the blue channel to any constant color. You can select the color using the **constantColor** controls.
alphaSource	i0	Select the input from which to take the alpha channel.
alphaChannel	A	Select what to use as the alpha channel: • **R** - use the red channel from **alphaSource**. • **G** - use the green channel from **alphaSource**. • **B** - use the blue channel from **alphaSource**. • **A** - use the alpha channel from **alphaSource**. • **Lum** - use the luminance from **alphaSource**. • **1** - set the alpha channel to white. • **0** - set the alpha channel to black. • **Const** - set the alpha channel to any constant color. You can select the color using the **constantColor** controls.
constantColor		
color	0.0000, 0.0000, 0.0000, 1.0000	The color (RGBA values) of the pixels in any channels that you have set to **Const**. You can also use the below RGB, HSL, or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.

ImageClamp

This node constrains, or clamps, values in the selected channels to a specified minimum and/or maximum range.

Connection Type	Connection Name	Function
Input	input	The image sequence the values of which you want to clamp.
	out_mask	An optional image to use as a mask. By default, the clamp effect is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageClamp parameters continued		
clamp	Both	Sets which values are use to clamp the input: • **Both** - **min** and **max** rgba values are clamped. • **Max** - only the **max** rgba clamps are used. • **Min** - only the **min** rgba clamps are used.
min		
red	0	Sets the minimum values at which the rgba channels are clamped. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
green	0	
blue	0	
alpha	0	

Control (UI)	Default Value	Function
max		
red	16	Sets the maximum values at which the rgba channels are clamped. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
green	16	
blue	16	
alpha	1	
inputs	Unpremultiplied	Select whether you are using a premultiplied or unpremultiplied input image: • **Premultiplied** - the ImageClamp node unpremultiplies the input, applies the clamp effect, and premultiplies the input again. This simulates applying the clamp before the premultiplication was done, as color corrections are typically applied on unpremultiplied images. • **Unpremultiplied** - the ImageClamp node simply applies the contrast change.

ImageContrast

This adjusts the input image's contrast around a fixed color point.

Connection Type	Connection Name	Function
Input	input	The image sequence the contrast of which you want to modify.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
contrast		
rgb	1	Adjusts the image contrast in the r, g, and b channels.
r	1	Adjusts the image contrast in the red channel only.
g	1	Adjusts the image contrast in the green channel only.
b	1	Adjusts the image contrast in the blue channel only.
a	1	Adjusts the image contrast in the alpha channel only.
fixedPoint		
fixedPoint	0.1800, 0.1800, 0.1800, 0.500	The point from which to influence the contrast. When contrast is greater than one, colors are moved away from this value, when the contrast is below one, colors are moved towards this value. You can also use the below RGB, HSL, or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
contrastFunction	Power	Select the contrast function to use: • **Power** • **Linear**
inputs	Unpremultiplied	Select whether you are using a premultiplied or unpremultiplied input image: • **Premultiplied** - the ImageContrast node unpremultiplies the input, applies the contrast change, and premultiplies the input again. This simulates applying the contrast change before the premultiplication was done, as color corrections are typically applied on unpremultiplied images. • **Unpremultiplied** - the ImageContrast node simply applies the contrast change.

ImageExposure

Allows you to adjust the exposure of the input sequence using f-stops or gain.

Connection Type	Connection Name	Function
Input	input	The image sequence the exposure of which you want to adjust.
	out_mask	An optional image to use as a mask. By default, the exposure effect is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
units: F-Stops		
fsIntensity	0	Adjusts f-stop intensity.
units: Gain		
gIntensity	1	Adjusts gain intensity.
units: F-Stops > fsColor		
red	0	Adjusts f-stop exposure in the red channel.
green	0	Adjusts f-stop exposure in the green channel.
blue	0	Adjusts f-stop exposure in the blue channel.
fsAlpha	0	Adjusts f-stop exposure in the alpha channel.
units: Gain > gColor		

Control (UI)	Default Value	Function
red	1	Adjusts gain exposure in the red channel.
green	1	Adjusts gain exposure in the green channel.
blue	1	Adjusts gain exposure in the blue channel.
gAlpha	1	Adjusts gain exposure in the alpha channel.
units	F-Stops	Select the units in which the exposure is altered: • **F-Stops** - use the **fsColor** controls to adjust exposure. • **Gain** - use the **gColor** controls to adjust exposure.

ImageFade

This node fades the input image to a color of your choosing. By default, the image is faded to black.

Connection Type	Connection Name	Function
Input	input	The image sequence that you want to fade to black (or a color of your choosing).
	out_mask	An optional image to use as a mask. By default, the fade effect is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageFade parameters continued		
amount	1	Dissolves between the **bg** image at 0 and the full merge effect at 1.

Control (UI)	Default Value	Function
fadeToColor		
fadeToColor	0.0000, 0.0000, 0.0000, 0.0000	The color (RGBA values) of the fade color. You can also use the below RGB, HSL, or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.

ImageGain

This node lets you adjust the gain in your input image. In other words, it multiplies a channel's values by a given factor, which has the effect of lightening the channel while preserving the blackpoint.

Connection Type	Connection Name	Function
Input	input	The image sequence the gain of which you wish to modify.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
units: F-Stops		
fsIntensity	0	Adjusts f-stop intensity.
units: Gain		
gIntensity	1	Adjusts gain intensity.

Control (UI)	Default Value	Function
units: F-Stops > fsColor		
red	0	Adjusts f-stop in the red channel.
green	0	Adjusts f-stop in the green channel.
blue	0	Adjusts f-stop in the blue channel.
fsAlpha	0	Adjusts f-stop in the alpha channel.
units: Gain > gColor		
red	1	Adjusts gain in the red channel.
green	1	Adjusts gain in the green channel.
blue	1	Adjusts gain in the blue channel.
gAlpha	1	Adjusts gain in the alpha channel.
units	F-Stops	Select the units in which the gain is altered: • **F-Stops** - use the **fsColor** controls to adjust gain. • **Gain** - use the **gColor** controls to adjust gain.

ImageGamma

Applies a constant gamma value to the selected channels. This lightens or darkens the mid-tones.

Connection Type	Connection Name	Function
Input	input	The image sequence the gamma of which you wish to modify.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
gamma		
rgb	1	Adjusts gamma in the red, green, and blue channels.
r	1	Adjusts gamma in the red channel.
g	1	Adjusts gamma in the green channel.
b	1	Adjusts gamma in the blue channel.
a	1	Adjusts gamma in the alpha channel.
fixedPoint		
fixedPoint	1.0000, 1.0000, 1.0000, 1.0000	The color (RGBA values). You can also use the below RGB, HSL, or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
inputs	Unpremultiplied	Select whether you are using a premultiplied or unpremultiplied input image: • **Premultiplied** - the ImageGamma node unpremultiplies the input, applies the gamma change, and premultiplies the input again. This simulates applying the gamma change before the premultiplication was done, as color corrections are typically applied on unpremultiplied images. • **Unpremultiplied** - the ImageGamma node simply applies the gamma change.

ImageInvert

Inverts a channel's values. To invert a channel is to subtract its values from 1, which causes its blacks to become white and its whites to become black. You may find this particularly useful to invert mattes.

Connection Type	Connection Name	Function
Input	input	The image sequence that values of which you want to invert.
	out_mask	An optional image to use as a mask. By default, the invert effect is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageInvert parameters continued		
mode	Additive	Sets the invert calculation mode: • Additive • Multiplicative
mode: Additive		
max	1.0000, 1.0000, 1.0000, 1.0000	The max color (RGBA values). You can also use the below RGB, HSL, or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.

ImageLevels

This node controls the input, gamma, and output levels of the input image.

Connection Type	Connection Name	Function
Input	input	The image sequence the levels of which you want to adjust.
	out_mask	An optional image to use as a mask. By default, the levels adjustment is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
inputMin		
inputMin	0.0000, 0.0000, 0.0000, 0.0000	Sets the minimum input level for the RGBA values. You can also use the HSL or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
inputMax		
inputMax	1.0000, 1.0000, 1.0000, 1.0000	Sets the maximum input level for the RGBA values. You can also use the HSL or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
gamma		
gamma	1.0000, 1.0000, 1.0000, 1.0000	Sets the gamma levels for the RGBA values. You can also use the HSL or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
outputMin		
outputMin	0.0000, 0.0000, 0.0000, 0.0000	Sets the minimum output level for the RGBA values. You can also use the HSL or HSV controls to set the color.

Control (UI)	Default Value	Function
		For more information, refer to the Color Widget Type in the Common Parameter Widgets.
outputMax		
outputMax	1.0000, 1.0000, 1.0000, 1.0000	Sets the maximum output level for the RGBA values. You can also use the HSL or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
direction	Forward	The transform direction, with **Forward** being used to bake in a color correction, and **Inverse** typically reversing out a previously-baked in correction.
clampMin	No	When set to **Yes**, levels are clamped to the specified **inputMin** and **outputMin** values.
clampMax	No	When set to **Yes**, levels are clamped to the specified **inputMax** and **outputMax** values.

ImageSaturation

This node is used to correct the input image's saturation (color intensity).

Connection Type	Connection Name	Function
Input	input	The image sequence the saturation of which you want to modify.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageSaturation parameters continued		
saturation	1	Controls overall image saturation. Values less than 1 reduce saturation, and the other way around.
coefficients		
red	0.2126	Adjusts the image in conjunction with the **saturation** control, but only affects the red channel.
green	0.7152	Adjusts the image in conjunction with the **saturation** control, but only affects the green channel.
blue	0.0722	Adjusts the image in conjunction with the **saturation** control, but only affects the blue channel.
normalize	enabled	When enabled, saturation calculations are normalized.

ImageThreshold

The ImageThreshold node sets the value of the output pixels of an image, based on whether the value is above or below the value in the **level** parameter.

Connection Type	Connection Name	Function
Input	input	The image sequence the output pixel values of which you want to set.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
level		
level	0.5000, 0.5000, 0.5000, 0.5000	For more information, refer to the Color Widget Type in the Common Parameter Widgets.
enableLow	Enabled	If enabled, any channel values below their corresponding level are set to the corresponding low value.
low		
low	0.0000, 0.0000, 0.0000, 0.0000	The output low value. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
enableHigh	Enabled	If enabled, any channel values greater than or equal to their corresponding level are set to the corresponding high value.
high		
high	1.0000, 1.0000, 1.0000, 1.0000	The output high value. For more information, refer to the Color Widget Type in the Common Parameter Widgets.

OCIOCDLTransform

This node applies an ASC CDL grade. The calculation uses **output = (i * s + o)^p** where **i** is the input value, **s** is **slope**, **o** is **offset** and **p** is **power**.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where the ASC CDL grade is applied.

Control (UI)	Default Value	Function
slope		
r	1	Adjusts the slope value in the red channel.
g	1	Adjusts the slope value in the green channel.
b	1	Adjusts the slope value in the blue channel.
offset		
r	0	Adjusts the offset value in the red channel.
g	0	Adjusts the offset value in the green channel.
b	0	Adjusts the offset value in the blue channel.
power		
r	1	Adjusts the power value in the red channel.
g	1	Adjusts the power value in the green channel.
b	1	Adjusts the power value in the blue channel.
saturation	1	Scales the image saturation using the 709 ASC primaries.
direction	forward	The direction of the transform, with **forward** being used to bake in the color transform, and **inverse** typically reversing out a previously-baked in transform.

OCIOColorSpace

This node converts the input colorspace to another specified colorspace.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where the colorspace conversion is applied.

Control (UI)	Default Value	Function
inColorSpace	linear	Sets the input colorspace to convert from.
outColorSpace	linear	Sets the output colorspace to convert to.
context Manually define keys and values for the OCIOColorspace node.		
key1; key2; key3; key4	N/A	Set a key that corresponds to the value of the same number in order to modify elements of the OCIO node, for example, setting a key called SHOT now tells the node that the corresponding value should be used for shots.
value1; value2; value3; value4	N/A	Set a value that corresponds to the key of the same number in order to modify elements of the OCIO node, for example, setting a value fora key called SHOT now tells the node that this value should be used in this context.

OCIODisplay

This node is used to convert the input colorspace to display device-suitable values.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where the colorspace conversion is applied.

Control (UI)	Default Value	Function
inputColorSpace	linear	Sets the input colorspace to convert from.
display	sRGB	Sets the output display colorspace to convert to.
view	sRGB	Specifies the colorspace transform to apply to the scene or image, from the options **Film**, **Log**, or **Raw**.
exposure		
rgb	0	Sets the exposure level for the r, g, and b channels together.
r	0	Sets the exposure level for the red channel.
g	0	Sets the exposure level for the green channel.
b	0	Sets the exposure level for the blue channel.
context Manually define keys and values for the OCIODisplay node.		
key1; key2; key3; key4	N/A	Set a key that corresponds to the value of the same number in order to modify elements of the OCIO node, for example, setting a key called SHOT now tells the node that the corresponding value should be used for shots.
value1; value2; value3; value4	N/A	Set a value that corresponds to the key of the same number in order to modify elements of the OCIO node, for example, setting a value fora key called SHOT now tells the node that this value should be used in this context.

OCIOFileTransform

This node applies a LUT transform using a specified file.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where the LUT transform is applied.

Control (UI)	Default Value	Function
src	N/A	Specifies the src file path and name to use for the transform. This can be any file format that OpenColorIO supports: **.3dl**, **.cc**, **.ccc**, **.csp**, **.cub**, **.cube**, **.lut** (Houdini), **.mga**, **.m3d**, **.spi1d**, **.spi3d**, **.spimtx**, **.vf**
cccid	N/A	When **src** points to a **.ccc** file, specify the id to lookup. OpenColorIO::Contexts (envvars) are obeyed.
direction	forward	The direction of the transform, with **forward** being used to bake in a LUT transform, and **inverse** typically reversing out a previously-baked in LUT transform.
interpolation	linear	Specifies the interpolation method. This is ignored if the file used is not a LUT. The following interpolation methods are listed from fastest to most accurate: • **nearest** • **linear** • **tetrahedral** • **best**
context Manually define keys and values for the OCIOFileTransform node.		
key1; key2; key3; key4	N/A	Set a key that corresponds to the value of the same number in order to modify elements of the OCIO node, for example, setting a key called SHOT now tells the node that the corresponding value should be used for shots.
value1; value2; value3; value4	N/A	Set a value that corresponds to the key of the same number in order to modify elements of the OCIO node, for example, setting a value fora key called SHOT now tells the node that this value should be used in this context.

OCIOLogConvert

This node can be used to override the Kodak-recommended settings when making Cineon conversions in either direction (lin to log or log to lin). It's rare that you would want to override these settings, but if it becomes necessary you can use the OCIOLogConvert node. If you do, you should also check **rawData** in the ImageRead and ImageWrite node controls to skip the automatic conversion.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want the log to lin or lin to log conversion to be applied.

Control (UI)	Default Value	Function
operation	Lin To Log	Select the operation to perform: • **Log To Lin** - convert from a logarithmic (Cineon) format to Katana's linear colorspace. • **Lin To Log** - convert from Katana's linear colorspace to a logarithmic (Cineon) format.

OCIOLookTransform

This node provides a way to apply per-shot color correction as specified using the OpenColorIO look mechanism.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where the color correction is applied.

Control (UI)	Default Value	Function
look	N/A	Sets which looks to apply, referencing the OCIO configuration. You can chain looks together using a list delimited by commas or colons. To indicate direction, you can also use the **+** and **-** modifiers.
direction	forward	The direction of the transform, with **forward** being used to bake in a transform, and **inverse** typically reversing out a previously-baked in transform.
inColorSpace	linear	Sets the input colorspace to convert from.
outColorSpace	linear	Sets the output colorspace to convert to.
ignoreErrors	disabled	When enabled, a missing OpenColorIO look forces this fail. When disabled, a missing OpenColorIO look is treated as a normal colorspace conversion.
context Manually define keys and values for the OCIOLookTransform node.		
key1; key2; key3; key4	N/A	Set a key that corresponds to the value of the same number in order to modify elements of the OCIO node, for example, setting a key called SHOT now tells the node that the corresponding value should be used for shots.
value1; value2; value3; value4	N/A	Set a value that corresponds to the key of the same number in order to modify elements of the OCIO node, for example, setting a value fora key called SHOT now tells the node that this value should be used in this context.

Composite Nodes

The following section describes Katana's 2D **Composite** nodes.

ImageIn

This node layers images together using the **In** compositing algorithm: **Bf**. It only shows the areas of the background that overlap with the alpha of the foreground. It can be useful for combining mattes.

You can also specify a different compositing algorithm using the **operation** control.

Connection Type	Connection Name	Function
Input	bg	The background image.
	fg	The foreground image.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageIn parameters continued		
operation	In	If you don't want layer the images together using the **In** compositing operation, select the operation to use instead. The following conventions apply to the below operation descriptions: • **F** refers to the **fg** input. • **f** refers to the **fg** input's alpha channel. • **B** refers to the **bg** input. • **b** refers to the **bg** input's alpha channel.
operation (continued)		The available operations (based on the Porter & Duff paper) are:

Control (UI)	Default Value	Function
		• **Atop** - Fb+B(1-f). This shows the shape of the background, with the foreground covering the background where the images overlap. • **Average** - (F+B)/2. This produces the average of the two images. The result is darker than the original images, but accentuates highlights. • **Difference** - abs(F-B). This shows how much the pixels differ and is useful for comparing two very similar images. • **Divide** - B/F. This divides the background values by the foreground values. • **Exclusion** - F+B-2FB. This is a more photographic form of **Difference**. • **From** - F-B. This subtracts the background from the foreground. For subtracting the foreground from the background instead, see **Minus**. • **Geometric** - 2FB/(F+B). This is another way of averaging two images. Visually, it's close to **Min**. • **Hypot** - sqrt(F*F+B*B). This resembles the **Add** and **Screen** operations. The result is not as bright as **Add**, but brighter than **Screen**. **Hypot** works with values above 1. It can be is useful for adding reflections, as an alternative to **Screen**. • **In** - Bf. This only shows the areas of the background that overlap with the alpha of the foreground. It can be useful for combining mattes. • **Matte** - Ff*B(1-f). This is a premultiplied **Over**. Use unpremultiplied images with this operation. • **Max** - max(F,B). This takes the maximum values of both images. This is a good way to combine mattes and useful for bringing aspects like bright hair detail through.
operation (continued)		• **Min** - min(F,B). This takes the minimum values of both images. • **Minus** - B-F. This subtracts the foreground from the background. For subtracting the background from the

Control (UI)	Default Value	Function
		foreground instead, see **From**. • **Multiply** - FB. This multiplies the values of the foreground by the values of the background. It can be used to composite darker values from the foreground with the background image - dark gray smoke shot against a white background, for example. • **Out** - B(1-f). This only shows the areas of the background that do not overlap with the alpha of the foreground. This can be useful for combining mattes. • **Over** - F+B(1-f). This layers the foreground over the background according to the alpha of the foreground. This is the most commonly used operation. It's used when layering a foreground element over a background plate. • **Plus** - F+B. This produces the sum of the foreground and background. Note that the add algorithm may result in pixel values higher than 1.0. • **Screen** - F+B-FB. This is similar to **Hypot**, but clamps pixel values to 1.0. This is mostly useful for combining mattes. • **Under** - F(1-b)+B. This is the reverse of the **Over** operation. It layers the background over the foreground according to the alpha of the background.
amount	1	Dissolves between the **bg** image at 0 and the full merge effect at 1.
displayWindow	Background	The frame size to output in the event that the **fg** and **bg** inputs are different sizes: • **Background** - output the frame size of the **bg** input. • **Foreground** - output the frame size of the **fg** input. • **Union** - output a combination of the **bg** and **fg** inputs' frame sizes. • **Intersection** - output an intersection of the **bg** and **fg** inputs' frame sizes. This restricts the output to the area where the two frame sizes overlap.
clampAlpha	enabled	When enabled, the output alpha channel is clamped to the 0-1 range. Color channels (RGB) are not affected.

ImageMerge

This node is a generic merge node that is able to perform all the other merge operations supported by Katana.

Connection Type	Connection Name	Function
Input	bg	The background image.
	fg	The foreground image.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageMerge parameters continued		
operation	Merge	If you don't want layer the images together using the **Merge** compositing operation, select the operation to use instead. The following conventions apply to the below operation descriptions: • **F** refers to the **fg** input. • **f** refers to the **fg** input's alpha channel. • **B** refers to the **bg** input. • **b** refers to the **bg** input's alpha channel.
operation (continued)		The available operations (based on the Porter & Duff paper) are:

Control (UI)	Default Value	Function
		• **Atop** - Fb+B(1-f). This shows the shape of the background, with the foreground covering the background where the images overlap. • **Average** - (F+B)/2. This produces the average of the two images. The result is darker than the original images, but accentuates highlights. • **Difference** - abs(F-B). This shows how much the pixels differ and is useful for comparing two very similar images. • **Divide** - B/F. This divides the background values by the foreground values. • **Exclusion** - F+B-2FB. This is a more photographic form of **Difference**. • **From** - F-B. This subtracts the background from the foreground. For subtracting the foreground from the background instead, see **Minus**. • **Geometric** - 2FB/(F+B). This is another way of averaging two images. Visually, it's close to **Min**. • **Hypot** - sqrt(F*F+B*B). This resembles the **Add** and **Screen** operations. The result is not as bright as **Add**, but brighter than **Screen**. **Hypot** works with values above 1. It can be is useful for adding reflections, as an alternative to **Screen**. • **In** - Bf. This only shows the areas of the background that overlap with the alpha of the foreground. It can be useful for combining mattes. • **Matte** - Ff*B(1-f). This is a premultiplied **Over**. Use unpremultiplied images with this operation. • **Max** - max(F,B). This takes the maximum values of both images. This is a good way to combine mattes and useful for bringing aspects like bright hair detail through.
operation (continued)		• **Min** - min(F,B). This takes the minimum values of both images. • **Minus** - B-F. This subtracts the foreground from the background. For subtracting the background from the

Control (UI)	Default Value	Function
		foreground instead, see **From**. • **Multiply** - FB. This multiplies the values of the foreground by the values of the background. It can be used to composite darker values from the foreground with the background image - dark gray smoke shot against a white background, for example. • **Out** - B(1-f). This only shows the areas of the background that do not overlap with the alpha of the foreground. This can be useful for combining mattes. • **Over** - F+B(1-f). This layers the foreground over the background according to the alpha of the foreground. This is the most commonly used operation. It's used when layering a foreground element over a background plate. • **Plus** - F+B. This produces the sum of the foreground and background. Note that the add algorithm may result in pixel values higher than 1.0. • **Screen** - F+B-FB. This is similar to **Hypot**, but clamps pixel values to 1.0. This is mostly useful for combining mattes. • **Under** - F(1-b)+B. This is the reverse of the **Over** operation. It layers the background over the foreground according to the alpha of the background.
amount	1	Dissolves between the **bg** image at 0 and the full merge effect at 1.
displayWindow	Background	The frame size to output in the event that the **fg** and **bg** inputs are different sizes: • **Background** - output the frame size of the **bg** input. • **Foreground** - output the frame size of the **fg** input. • **Union** - output a combination of the **bg** and **fg** inputs' frame sizes. • **Intersection** - output an intersection of the **bg** and **fg** inputs' frame sizes. This restricts the output to the area where the two frame sizes overlap.
clampAlpha	enabled	When enabled, the output alpha channel is clamped to the 0-1 range. Color channels (RGB) are not affected.

ImageOut

This node layers images together using the **Out** compositing algorithm: **B(1-f)**. Only shows the areas of the background that do not overlap with the alpha of the foreground. This can be useful for combining mattes.

Connection Type	Connection Name	Function
Input	bg	The background image.
	fg	The foreground image.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageOut parameters continued		
operation	Out	If you don't want layer the images together using the **Out** compositing operation, select the operation to use instead. The following conventions apply to the below operation descriptions: • **F** refers to the **fg** input. • **f** refers to the **fg** input's alpha channel. • **B** refers to the **bg** input. • **b** refers to the **bg** input's alpha channel.
operation (continued)		The available operations (based on the Porter & Duff paper) are: • **Atop** - Fb+B(1-f). This shows the shape of the

Control (UI)	Default Value	Function
		background, with the foreground covering the background where the images overlap. • **Average** - (F+B)/2. This produces the average of the two images. The result is darker than the original images, but accentuates highlights. • **Difference** - abs(F-B). This shows how much the pixels differ and is useful for comparing two very similar images. • **Divide** - B/F. This divides the background values by the foreground values. • **Exclusion** - F+B-2FB. This is a more photographic form of **Difference**. • **From** - F-B. This subtracts the background from the foreground. For subtracting the foreground from the background instead, see **Minus**. • **Geometric** - 2FB/(F+B). This is another way of averaging two images. Visually, it's close to **Min**. • **Hypot** - sqrt(F*F+B*B). This resembles the **Add** and **Screen** operations. The result is not as bright as **Add**, but brighter than **Screen**. **Hypot** works with values above 1. It can be is useful for adding reflections, as an alternative to **Screen**. • **In** - Bf. This only shows the areas of the background that overlap with the alpha of the foreground. It can be useful for combining mattes. • **Matte** - Ff*B(1-f). This is a premultiplied **Over**. Use unpremultiplied images with this operation. • **Max** - max(F,B). This takes the maximum values of both images. This is a good way to combine mattes and useful for bringing aspects like bright hair detail through.
operation (continued)		• **Min** - min(F,B). This takes the minimum values of both images. • **Minus** - B-F. This subtracts the foreground from the background. For subtracting the background from the foreground instead, see **From**.

Control (UI)	Default Value	Function
		• **Multiply** - FB. This multiplies the values of the foreground by the values of the background. It can be used to composite darker values from the foreground with the background image - dark gray smoke shot against a white background, for example. • **Out** - B(1-f). This only shows the areas of the background that do not overlap with the alpha of the foreground. This can be useful for combining mattes. • **Over** - F+B(1-f). This layers the foreground over the background according to the alpha of the foreground. This is the most commonly used operation. It's used when layering a foreground element over a background plate. • **Plus** - F+B. This produces the sum of the foreground and background. Note that the add algorithm may result in pixel values higher than 1.0. • **Screen** - F+B-FB. This is similar to **Hypot**, but clamps pixel values to 1.0. This is mostly useful for combining mattes. • **Under** - F(1-b)+B. This is the reverse of the **Over** operation. It layers the background over the foreground according to the alpha of the background.
amount	1	Dissolves between the **bg** image at 0 and the full merge effect at 1.
displayWindow	Background	The frame size to output in the event that the **fg** and **bg** inputs are different sizes: • **Background** - output the frame size of the **bg** input. • **Foreground** - output the frame size of the **fg** input. • **Union** - output a combination of the **bg** and **fg** inputs' frame sizes. • **Intersection** - output an intersection of the **bg** and **fg** inputs' frame sizes. This restricts the output to the area where the two frame sizes overlap.
clampAlpha	enabled	When enabled, the output alpha channel is clamped to the 0-1 range. Color channels (RGB) are not affected.

ImagePremultiply

This node premultiplies (mult) the rgb channels by the **alphaChannel** when an image is connected to the **alpha** input. Otherwise, **a** is read from the **input** leaving the alpha channel unchanged.

Also see ImageUnpremultiply.

Connection Type	Connection Name	Function
Input	input	The image sequence you want to premultiply. If no **alpha** input is connected, this input should contain an alpha channel to use for premultiplying the color channels.
	alpha	An optional input for attaching a separate channel that is used to premultiply the **input** image. If this input is connected, you can use **alphaChannel** to select the channel to use.
	out_mask	An optional image to use as a mask. By default, the premultiplication is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImagePremultiply parameters continued		
alphaChannel	A	If a separate **alpha** input is provided, choose which of its channels to use to premultiply. If no separate **alpha** input is provided, the alpha from **input** is used and this control is disabled.

ImageUnpremultiply

This node divides the RGB channels by the **alphaChannel** when an image is connected to the **alpha** input. Otherwise, **a** is read from the **input** leaving the alpha channel unchanged.

Also see ImagePremultiply.

Connection Type	Connection Name	Function
Input	input	The image sequence you want to unpremultiply. If no **alpha** input is connected, this input should contain an alpha channel to use for unpremultiplying the color channels.
	alpha	An optional input for attaching a separate channel that is used to unpremultiply the **input** image. If this input is connected, you can use **alphaChannel** to select the channel to use.
	out_mask	An optional image to use as a mask. By default, the unpremultiplication is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageUnpremultiply parameters continued		
alphaChannel	A	If a separate **alpha** input is provided, choose which of its channels to use to unpremultiply. If no separate **alpha** input is provided, the alpha from **input** is used and this control is disabled.

ImageZMerge

The ImageZMerge node applies a simple A over B composite. A and B are determined by examining the corresponding depth image and making A the image that has the lower (closer) depth value. A depth of **0** is assumed to be infinitely far away. An accumulated depth for the result is also available as the second output. Inputs are ordered as follows: Image1, Depth1, Image2, Depth2.

Connection Type	Connection Name	Function
Input	i0	The input ports you want to set for different parts of the node graph. You can add as many numbered input ports as you want by pressing ▼ in the node.
	i1	
	i2	
	i3	

Control (UI)	Default Value	Function
depthChannel 	N/A	Specifies the channel (R, G, B, or A) that contains the depth values in the depth images.

Filter Nodes

The following section describes Katana's 2D **Filter** nodes.

ImageBlur

Adds blur to an image or matte using Box, Triangle, Gaussian, Bell, BSpline, or Mitchell filter algorithms. The blur value is calculated for image pixels by examining their neighbors within the constraints of the **xAmount** and **yAmount** controls, and applying the selected algorithms.

Connection Type	Connection Name	Function
Input	input	The image sequence you want to blur.
	out_mask	An optional image to use as a mask. By default, the blur effect is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageBlur parameters continued		
xAmount	0	Sets the horizontal radius (in pixels) within which pixels are compared to calculate the blur. Higher values widen the compare area, producing more blur.
yAmount	xAmount	Sets the vertical radius (in pixels) within which pixels are compared to calculate the blur. Higher values widen the compare area, producing more blur. By default, this value is the same as **xAmount**.
filter	Gaussian	Select the filtering algorithm to use: • **Box** • **Triangle** • **Gaussian** • **Bell** • **BSpline** • **Mitchell**
borderExtend	Clamp	Select the border extend method for pixels required beyond the image borders:

Control (UI)	Default Value	Function
		• **Mirror** • **Clamp** • **Background**
useOverscan	globals.compDefaults.useOverscan	Sets whether to use upstream overscan (if available) during the border extension process. If overscan is available (and of usable quality), this typically yields superior results around frame edges. However, if you are unsure of this procedure or the integrity of overscanned areas is unknown, it's safer to leave this disabled. Overscan refers to image pixel data outside of the displayWindow and can be inspected using options in the Monitor. For information on explicitly manipulating these regions, see the ImageCrop node.
channelAmounts		
red	1	Applies a multiplier to the blur amount for the red channel.
green	1	Applies a multiplier to the blur amount for the green channel.
blue	1	Applies a multiplier to the blur amount for the blue channel.
alpha	1	Applies a multiplier to the blur amount for the alpha channel.

I/O Nodes

The following section describes Katana's 2D **I/O** nodes.

ImageRead

This node loads images from disk, using the native resolution and the frame range for the sequence. It converts all imported sequences to Katana's linear colorspace automatically, but there are options to control this. Note that Katana's image processing operations are written assuming they are working on linear images, so be careful if you change the default input colorspace conversion. All of Katana's image processing is implemented in floating point, so files are converted to float at input.

Control (UI)	Default Value	Function
file	N/A	The image sequence to load. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
image		
rawData	disabled	When enabled, Katana skips the automatic colorspace conversion. Note that Katana is inherently a floating-point system. Thus, if integer data is loaded (at any bit-depth), the pixels are mapped to the range of [0, 1].
colorspace	auto	Select the colorspace for the file on disk. Upon load, the image sequence is converted from this colorspace to Katana's native floating-point linear colorspace. The default value, **auto**, means Katana tries to determine the bit depth from the file header and the colorspace from the file name. If Katana gets this wrong or the file is not named in a standard way, you can use this control to force Katana to assume the image data is in the selected colorspace and bit depth.

Control (UI)	Default Value	Function
		You can also use this control to avoid the colorspace conversion entirely by specifying **lnzf** or **ncf**, which indicate that the file is already linear. Bear in mind, however, that most image processing operations in Katana presume linear input data. The results of image processing operations in Katana are not defined, tested, or supported for non-linear image data. All operators have been implemented assuming input images are linear. **Note:** This option only appears when **rawData** is disabled.
isProxy	disabled	When enabled, Katana assumes the loaded image sequence is a proxy rather than a full-resolution image. This is preferable to manually resizing the image, as it is more efficient when proxy-rendering is enabled.
image > isProxy: enabled > fullResFrame		
[resolution]	Dependent on Project Settings	When **isProxy** is enabled, you can use this control to select the resolution for the full-resolution image.
left	timing.missingFrameBounds.left	Sets the left position of the rectangle.
bottom	timing.missingFrameBounds.bottom	Sets the bottom position of the rectangle.
width	timing.missingFrameBounds.width	Sets the width of the rectangle.
height	timing.missingFrameBounds.height	Sets the height of the rectangle.
timing		

Control (UI)	Default Value	Function
frame	frame	Sets the frame number actually read from disk prior to applying the **inMode**, **outMode**, **firstFrame**, and **lastFrame** settings. When a downstream node requests an image from an ImageRead node, ImageRead evaluates this control to determine the frame number to read from disk (by default this is the current time). The result is compared against the **firstFrame** and **lastFrame** values and, if necessary, any remapping of the actual frame number is done based on the **inMode** and **outMode** settings. You can retime or offset your input by using an expression or a curve here, but note that currently Katana only reads the nearest frame and doesn't generate in-between frames (no optical flow interpolation). The value is forced to an integer at the time it's used, so you don't need to worry about this if you don't want to.
inMode	Black	Sets what to do when a frame is required at a time value prior to **firstFrame**: • **Black**firstFrame to black. • **Freeze** • **Repeat** • **Mirror**
outMode	Black	Sets what to do when a frame is required at a time value after **lastFrame**: • **Black** -lastFrame to black.

Control (UI)	Default Value	Function
		• **Freeze** • **Repeat** • **Mirror**
firstFrame	globals.inTime	Sets the first valid frame of the sequence of images on disk. If a frame prior to **firstFrame** is required, its contents are determined based on **inMode**. If the **file** control has frame range values in it and this control is left at its default value, the value in the **file** control is obeyed.
lastFrame	globals.outTime	Sets the last valid frame of the sequence of images on disk. If a frame beyond **lastFrame** is required, its contents are determined based on **outMode**. If the **file** control has frame range values in it and this control is left at its default value, the value in the **file** control is obeyed.
lockSettings	disabled	When enabled, the **firstFrame**, **lastFrame**, **inMode**, and **outMode** values aren't automatically updated when a new file sequence is chosen.
missingFrames	Error	Specifies what to do if a frame is not found: • **Error** - have the render fail with an error. • **Black** - replace any missing frames with black. • **Nearest** - replace any missing

Control (UI)	Default Value	Function
		frames with the nearest frame. • **Checkerboard** - replace any missing frames with a checkerboard image. **Note:** If no frames in the image sequence are present, the render fails regardless of this control's setting.
advanced		
includeInErrorChecking	enabled	When enabled, Katana includes this node when it automatically checks ImageRead nodes for errors. When disabled, Katana excludes this node when it automatically checks ImageRead nodes for errors.

ImageWrite

ImageWrite writes its incoming image to a file on disk. The ImageWrite node, unless defaults are overridden, converts images from Katana's linear colorspace to the colorspace named in the filename. Katana image processing nodes work entirely in floating point, so images are also converted from floating point to the bit depth specified in the options for the format.

ImageWrite contains controls (**channels** and **outputFrame**) to force the output regardless of what may be coming into the node. However, if the output format cannot support the settings (for example, **.jpeg** doesn't support an alpha channel), the extra information is discarded.

The ImageWrite node supports the following file formats: **.exr**, **.rla**, **.cin**, **.png**, **.tif**, **.tiff**, **.jpg**, **.jpeg**, **.dpx**, and **.hist**.

Connection Type	Connection Name	Function
Input	in	The incoming image that you want written to a file on disk.

Control (UI)	Default Value	Function
passName	comp	Sets the name used in the directories generated for this ImageWrite node. The **passName** should be unique for each ImageWrite node in the scene.
activeViews	main	Determines which views generate output images when hot-rendered or batch-rendered • **Enable All** - All views generate output images. • **main** - Only the main view generates output images. • **left** - Only the left view generates output images. • **right** - Only the right view generates output images.
singleFrame	disabled	When enabled, Katana only renders a single frame (for example, image_res.0001.exr) rather than an image sequence (for example, image_res.#.exr). You can specify the frame number using the **frame** control below. This also produces a render error when rendering on any frame other than the specified frame.
singleFrame: enabled		
frame	globals.inTime	Sets the frame to render when **singleFrame** is enabled.
inputs		
[identifier]	N/A	Defines short input identifiers. The identifier is included in the input/output input names and is used as a prefix for the output asset rep. %V is replaced with the view name. %v is replaced with the appropriate asset token.

Control (UI)	Default Value	Function
		Note: Input identifiers have no effect on file names, only assets.
mode	file	Sets whether to write a file or define a dependency: • **file** • **dependency**
file	N/A	Sets the file path and name for the rendered image (s). **Note:** If **mode** is set to **dependency**, this control is hidden.
inputs > mode: file > image		
proxyOnCue	enabled	
channels	Input	Selects the channels to render: • **RGBA** - Render the red, green, blue, and alpha channels. If any of the color channels are missing from the input, they are filled with 0 (pure black). If the alpha channel is missing, it is filled with 1 (pure white or fully opaque). • **RGB** - Render the red, green, and blue channels. If any of these channels are missing from the input, they are filled with 0 (pure black). • **A** - Only render the alpha channel. If this channel is missing from the input, it is filled with 1 (pure white or fully opaque). • **Input** - Render all channels that exist in the input. If the file format does not support the input channel configuration, required but missing channels are filled with 0 (color channels) or 1 (alpha).
rawData	disabled	When enabled, Katana skips the automatic colorspace conversion (that is, the conversion from

Control (UI)	Default Value	Function
		its native linear floating-point format to the output colorspace).
colorspace	linear	Katana converts from linear to this colorspace when writing the file to disk. The default value, **auto**, means Katana tries to determine the output colorspace from the file name.
colorConvert	enabled	When enabled, Katana converts rendered image data from its native linear colorspace to the output colorspace specified in the file name. This is desirable in nearly every situation. A case where you would want to set this to disabled is if you know the data being rendered is in a colorspace other than linear (such as the re-projection of a log plate) and you want to name the output file log without a linear to log conversion.
fileFormat	exr	Sets the file format to output: • **auto** - Katana tries to determine the output format from the file name. • **exr** • **rla** • **cin** • **png** • **tif** • **tiff** • **jpg** • **jpeg** • **dpx** • **hist**
inputs > mode: file > image > fileFormat: exr		
exrCompression	Wavelet	Defines the exr compression method to use. All methods are lossless (with the exception of **Pixar 24**,

Control (UI)	Default Value	Function
		which is lossless but restricts the pixels to 24-bit float). **Wavelet** is generally preferable as it offers ~2:1 compression even on grainy data.
exrBitDepth	16	Sets the floating point precision of the rendered exr file: • **16** - half float. This is recommended for all color passes. • **32** - full float. This is recommended for all ncf data arbitrary output variables (AOVs).
exrType	Tiled	Sets whether the exr file is written to support: • **Tiled** - random tile access. • **Scanline** - random scanline access.
comments	N/A	Optional field for any comments you want to store in the output file's comment metadata field. Currently, this is only supported on the exr file format.
inputs > mode: file > image > fileFormat: exr > exrType: Tiled		
exrTileWidth	256	Sets the tile width to use when writing to tiled exr files.
exrTileHeight	256	Sets the tile height to use when writing to tiled exr files.
exrTileWorldAlign	disabled	When enabled (in conjunction with shrinkwrapping), the data rectangle is adjusted (top+left) so that the internal tile boundaries are aligned with world coordinates. This improves memory usage / performance for programs that process image tiles (such as Katana). **Note:** This does not guarantee that tiles are aligned - it merely attempts to meet this condition.

Control (UI)	Default Value	Function
inputs > mode: file > image > fileFormat: rla		
rlaBitDepth	auto	Sets the bit depth of the rendered file. The default value, **auto**, means Katana tries to determine the bit depth from the colorspace. The other options are: • **8-bit** • **10-bit** • **16-bit** • **32-bit**
inputs > mode: file > image > fileFormat: png		
pngBitDepth	auto	Sets the bit depth of the rendered file. The default value, **auto**, means Katana tries to determine the bit depth from the colorspace. The other options are: • **8-bit** • **16-bit**
inputs > mode: file > image > fileFormat: tif or tiff		
tifCompression	LZW	The tiff compression method to use: • **None** - No compression method is used. • **LZW** - The LZW compression method is used. This is lossless, so it is usually preferable to use it unless there is an issue with compatibility in the target reader.
tifBitDepth	auto	The bit depth of the rendered file. The default value, **auto**, means Katana tries to determine the bit depth from the colorspace.The other options are: • **8-bit** • **16-bit** • **32-bit**
tifPredictor	None	The predictor type to use when **tifCompression** is enabled: • **None** - No prediction is used.

Control (UI)	Default Value	Function
		• **Horizontal** - Horizontal prediction is used. This can result in smaller file sizes, but may present compatibility issues for some programs, such as MAXON's Cinema4D.
inputs > mode: file > image > fileFormat: jpg or jpeg		
jpgQuality	100	The quality to use when generating the jpg file. Higher values generate larger file sizes, with 100 representing the best quality image and 0 representing the lowest.
inputs > mode: file > bounds		
displayWindow	input	The frame size to write to the file: • **input** - Use the frame size from the input. This crops off image data outside the frame or pads the frame with black if the image bounds do not fill the frame already. • **manual** - Crop the output to the specified frame size, padding with black if necessary.
dataWindow	shrinkwrap	The image area to write to the file: • **shrinkwrap** - Make sure the area is no larger than the frame size. This is the typical choice. If the format supports separate data and display windows (for example, the exr format does), the data window is clipped to the frame. • **displayWindow** - Write whatever area the input image data window covers (even if it exceeds the frame size). This only works with formats like exr that support a data window different from the display window. This is useful for writing out overscan images where the data extends beyond the frame. • **manual** - Crop the image area to the specified size, padding with black if necessary.

Control (UI)	Default Value	Function
		Note: Make sure other applications you are using support the selection you make. For example, if you select **displayWindow**, any other applications that read the output need to be able to handle separate data and display windows. You also need to use a format (like exr) that supports the concept, otherwise the data window is still clipped to the frame.
inputs > mode: file > bounds > overscan		
left	0	Overscan specifies the number of pixels to pad the render request in each direction during a disk render (including batch renders). The display window is unchanged, but this expands the data window to include any extra input data that has been made available by the expanded render request. Note that if **dataWindow** is set to **shrinkwrap**, the data window is still shrunk inward to encompass only the non-zero pixels in the image. Overscan simply enlarges the area that is initially rendered and under consideration for shrinkwrapping. Overscan has no effect when **dataWindow** is set to **manual**. You must include the desired overscan amount directly in the manual data window that you set.
bottom	0	
right	0	
top	0	
inputs > mode: file > bounds > displayWindow: manual		
displayWIndowResolution	512sq	Sets the resolution of the display window using the dropdown menu. This is a useful override if there exists different resolution names with the same resolution width and height.

Control (UI)	Default Value	Function
width	512	Defines the display window resolution manually.
height	512	
postScripts > Add		
Add Post Script	N/A	Allows you to add post script commands.
farmSettings		
setActiveFrameRange	disabled	Sets how the active frame range for rendering is defined: When enabled, the **activeFrameRange** controls are displayed, which define the active frame range for rendering. When disabled, Katana assumes that the active frame range is the same as the range between globals.inTime and globals.outTime. These settings affect outline file generation and guarantee that even if the node is called to render, it only writes files for frames in the active range.
farmFileName	N/A	Defines the farm file name and path.
versionUp	Auto	Sets whether the outputs of this node are versioned up when rendered on the queue: • **Auto** - use the global setting specified in the outline file. • **Yes** - outputs version up. • **No** - outputs don't version up.
threadable	enabled	Determines whether the queue is allowed to assign multiple cores to a frame of this render. When enabled, the queue may optionally thread the render. When disabled, the queue must use only one core.

Control (UI)	Default Value	Function
memory	N/A	Sets the memory requirement for the farm layer. Memory can be defined as **m** for megabyte or **g** for gigabyte. For example, 512m or 2g.
excludeFromFarmOutput Generation	disabled	When enabled, this node does not appear in any generated farm file (however, the node is still renderable if called directly). Enabling this control hides the **forceFarmOutputGeneration** control.
forceFarmOutputGeneration	disabled	When enabled, this node always appears in generated farm files (regardless of whether it has any valid outputs). **Note:** If **excludeFromFarmOutputGeneration** is also enabled, the node does not appear in the generated farm file (**excludeFromFarmOutputGeneration** overrides **forceFarmOutputGeneration**).
farmSettings > setActiveFrameRange: enabled > activeFrameRange		
start	1	Sets the first frame in the active frame range when **setActiveFrameRange** is enabled.
end	1	Sets the last frame in the active frame range when **setActiveFrameRange** is enabled.

Source Nodes

The following section describes Katana's 2D **Source** nodes.

ImageCheckerboard

The ImageCheckerboard allows you to create a checkerboard pattern. You can specify the checkers' size and colors and the checkerboard's bounds.

Control (UI)	Default Value	Function
bounds		
[resolution]	Dependent on Project Settings	Select the size of the image.
▼	N/A	For more information, refer to the Rectangle Widget Type in the Common Parameter Widgets.
left	0	Set the left position of the ROI in the **Monitor** tab.
bottom	0	Set the bottom position of the ROI in the **Monitor** tab.
width	globals.width	Set the width of the ROI in the **Monitor** tab.
height	global.height	Set the height of the ROI in the **Monitor** tab.
color1		
color1	0.1000, 0.1000, 0.1000, 1.0000	The color (RGBA values) of the pixels for the first color in the checkerboard. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
color2		
color2	0.5000, 0.5000, 0.5000, 1.000	The color (RGBA values) of the pixels for the second color in the checkerboard. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
ImageCheckerboard parameters (Cont.)		
checkerSize	64.0, 64.0	Sets the size of the checkers under **width** and **height**.

ImageColor

Generates an image where every pixel is the same color. By default, the image is white.

Control (UI)	Default Value	Function
bounds		
[resolution]	Dependent on Project Settings	Select the size of the image.
bounds >	N/A	For more information, refer to the Rectangle Widget Type in the Common Parameter Widgets.
left	0	Lets you offset the image by adding this number of pixels to the left side of the image.
bottom	0	Lets you offset the image by adding this number of pixels below the image.
width	globals.width	The width of the image in pixels. The default setting, **globals.width**, resizes the image to the width of the **resolution** indicated on the **Project Settings** tab.
height	globals.height	The height of the image in pixels. The default setting, **globals.height**, resizes the image to the height of the **resolution** indicated on the **Project Settings** tab.
ImageColor parameters continued		
infiniteExtent	Disabled	When enabled, the color extends beyond the bounds.
color		
color	1.0, 1.0, 1.0, 1.0	The color (RGBA values) of every pixel in the image. You can also use the RGB, HSL, or HSV controls to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.

ImageRamp

Generates a gradation with various parameters.

Control (UI)	Default Value	Function
bounds		
[resolution]	Dependent on Project Settings	Sets the size of the display window using the dropdown menu.
bounds >	N/A	For more information, refer to the Rectangle Widget Type in the Common Parameter Widgets.
left	0	Set the left position of the ROI in the **Monitor** tab.
bottom	0	Set the bottom position of the ROI in the **Monitor** tab.
width	globals.width	Set the width of the ROI in the **Monitor** tab.
height	global.height	Set the height of the ROI in the **Monitor** tab.
ImageRamp parameters continued		
type	Horizontal	Set the pattern for the gradation from the options: • Horizontal • Vertical • Diagonal • Linear • Corner • Radial
interpolationColorspaces	linear	Specify the colorspace used for the color fields.
type: Horizontal or **Vertical**		
start > color	1.0000, 1.0000, 1.0000, 1.0000	The color (RGBA values) at the start of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type

Control (UI)	Default Value	Function
		in the Common Parameter Widgets.
end > color	0.0000, 0.0000, 0.0000, 0.0000	The color (RGBA values) at the end of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
type: Diagonal		
start > color	1.0000, 1.0000, 1.0000, 1.0000	The color (RGBA values) at the start of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
end > color	0.0000, 0.0000, 0.0000, 0.0000	The color (RGBA values) at the end of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
sense	Left to Right	Set whether the start color begins on the left and transitions to the end color on the right, or the other way around.
type: Linear		
start > color	1.0000, 1.0000, 1.0000, 1.0000	The color (RGBA values) at the start of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
start > Start	globals.width, globals.height	Set the position in the **Monitor** tab, or with global manipulators turned on, reposition the start point in the **Monitor** tab to adjust the values in the **Start** field.

Control (UI)	Default Value	Function
inner > add point	N/A	Add a point to the gradation, where a specific color can be set to a specific **position**. By default, when you add a point, the name is filled in the blank box with p_0, ascending in number as you add more points. This can be changed to a more meaningful name.
end > color	0.0000, 0.0000, 0.0000, 0.0000	The color (RGBA values) at the end of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
end > End	globals.width, globals.height	Set the position in the **Monitor** tab, or with global manipulators turned on, reposition the end point in the **Monitor** tab to adjust the values in the **End** field.
type: Corner		
bottomLeft	1.0000, 0.0000, 0.0000, 1.0000	The color (RGBA values) at the bottom-left corner of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
bottomRight	0.0000, 1.0000, 0.0000, 1.0000	The color (RGBA values) at the bottom-right corner of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
topRight	0.0000, 0.0000, 1.0000, 1.0000	The color (RGBA values) at the top-right corner of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
topLeft	0.0000, 0.0000,	The color (RGBA values) at the top-left corner of the

Control (UI)	Default Value	Function
	0.0000, 0.0000	ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
type: Radial		
start > color	1.0000, 1.0000, 1.0000, 1.0000	The color (RGBA values) at the start of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
start > Start	globals.width, globals.height	Set the position in the **Monitor** tab, or with global manipulators turned on, reposition the start point in the **Monitor** tab to adjust the values in the **Start** field.
end > color	0.0000, 0.0000, 0.0000, 0.0000	The color (RGBA values) at the end of the ImageRamp. Alternatively, you can also use the **color** options below to set the color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
radial > confineTo Circle	disabled	Set whether the start and end colors in the radial pattern are confined to a circle.
radial > radius	778	Specify the radius of the circle.
radial > aspectRatio	1	Specify the aspect ratio of the circle.
radial > innerRadius	0	Specify the radius of the inner circle, as set by the start color.
radial > fallOff	0.5	Controls the profile of the fall off between the start and end radii.

ImageText

An ImageText node is a type of 2D node that generates an image with text written into it.

Connection Type	Connection Name	Function
Input	scenegraph	A 3D scene from which attributes can be referenced in the text parameter of an ImageText node (see below).

Control (UI)	Default Value	Function
bounds		
[resolution]	Dependent on Project Settings	Select the size of the text frame.
left	0	Lets you offset the text frame by this number of pixels from the left.
bottom	0	Lets you offset the text frame by this number of pixels from the bottom.
width	globals.width	The width of the text frame in pixels. The default setting, **globals.width**, resizes the text frame to the width of the **resolution** indicated on the **Project Settings** tab. **Note:** You can only edit this field when the initial resolution is modified.
height	globals.height	The height of the text frame in pixels. The default setting, **globals.height**, resizes the text frame to the height of the **resolution** indicated on the **Project Settings** tab.

Control (UI)	Default Value	Function
		Note: You can only edit this field when the initial resolution is modified.
text	N/A	Enter the text you want to display here. You can optionally query scene graph values from an incoming 3D scene by: • Connecting a 3D scene as input • Creating a **text** GroupAttribute at **/root**, containing the attributes you are interested in using as children. • Reference attrs within the text node using the {attr:ATTRNAME} syntax.
fontSource	Builtin	Select: • **Builtin** - to use a built-in font (either Arial or Courier) for the text. • **File** - to use a font from an external font file for the text. Enter the file path to the font or use the file browser to browse to it. Fonts are loaded using FreeType2, which supports TrueType and OpenType fonts among others.
fontSource: Builtin		
font	Arial	Lets you select a font for the text when **fontSource** is set to **Builtin**: either **Arial** or **Courier**.
fontSource: File		
fontFile	N/A	Lets you select a font for the text when **fontSource** is set to **File**.
parameters continued		
size	18.0, size[0]	Sets the pixel size of the font. Note that because of the way fonts are generated from control splines that vary in size, you rarely get a character that is exactly this size.

Control (UI)	Default Value	Function
		No character ever renders larger than this size.
position	getDisplayWindow().width/2, getDisplayWindow().height/2	The pixel position at which the justified text is placed. For example, if you set **hjustify** to **Left** and **vjustify** to **Top**, the left side of the baseline of the first line of text is placed at this location. If you set **hjustify** to **Center** and **vjustify** to **Bottom**, the baseline of the last line of text is centered on this position horizontally.
hjustify	Center	Sets how to align the text horizontally: • **Left** - align the text along the left edge of the text frame, placing the left side of the text block at the location defined by **position**. This leaves the right edge of the text ragged. • **Center** - align the text from the center of the text frame, placing the center of the text block at the location defined by **position**. This leaves both edges of the text ragged. • **Right** - align the text along the right edge of the text frame, placing the right side of the text block at the location defined by **position**. This leaves the left edge of the text ragged.
vjustify	Center	Sets how to align the text vertically: • **Top** - align the text along the top edge of the text frame, placing the top baseline of the text block at the location defined by **position**. • **Center** - align the text from the center of the text frame, placing the center baseline of the text block at the location defined by **position**. • **Bottom** - align the text along the bottom of the text frame, placing the bottom baseline of the text block at the location defined by **position**. The baseline is the imaginary line upon which most letters

Control (UI)	Default Value	Function
		rest.
lineSpace	0	If you have several lines of text, this adjusts the spacing between each line. By using negative values, you can make the letters overlap.
wrapMode	None	Sets how to wrap long lines of text to fit inside the text frame: • **None** - long lines are not wrapped to fit inside the text frame. Some parts of the text may fall outside the frame and not be visible. • **Word** - long lines are split into several lines at word boundaries. • **Exact** - long lines are split into several lines at the closest point in the text that fits the text frame width, regardless of word boundaries.
wrapMode: Word or Exact		
wrapWidth	bounds.width	The width to use when calculating when to wrap the text.
color		
color	1.0, 1.0, 1.0, 1.0	The color (RGBA values) of the rendered text. You can also use the below RGB, HSL, or HSV controls to set the color of the text. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
parameters continued		
antiAliasingGamma	2.2	Gamma applied after text rasterization but before applying color. This affects anti-aliasing appearance.
missingAttributes	Ignore	Specifies behavior when missing 3D scene graph attributes are encountered: • **Ignore** - the missing attributes are ignored. • **Error** - the missing attributes cause a render error.

Transform Nodes

The following section describes Katana's 2D **Transform** nodes.

ImageCrop

This node removes, or crops, image information outside a defined area, though Katana has both a data window and a display window (to use the EXR terminology).

- The display window is the image frame.
- The data window is the area that actually contains pixels.



Note: The data window may be larger or smaller than the display window. If it is larger, image data exists that can be pulled into the frame by downstream operations. If smaller, savings in processing time and memory are achieved by not explicitly storing pixel values for all the constant color outside the useful image area.

Connection Type	Connection Name	Function
Input	input	The image sequence that you want to crop.

Control (UI)	Default Value	Function
bounds		
[resolution]	Dependent on Project Settings	Sets the size of the display window using the dropdown menu.
bounds >	N/A	For more information, refer to the Rectangle Widget Type in the Common Parameter Widgets.
left	0	Offset the display window by this number of pixels from the

Control (UI)	Default Value	Function
		left side of the data window.
bottom	0	Offset the display window by this number of pixels from the bottom side of the data window.
width	globals.width	Adjusts the width of the display window in pixels.
height	globals.height	Adjusts the height of the display window in pixels.
ImageCrop parameters continued		
reformat	disabled	When enabled, reposition the cropped area to the origin and changes the display window.
reformat: enabled		
allowOverscan	disabled	This allows the node to generate overscan (if possible). Overscan refers to image pixel data outside of the display window and can be inspected using options in the Monitor.

ImageOrient

This node allows you to rotate, flip, and flop the input image around its center. A flip on the x axis mirrors the image vertically. A flop on the on the y axis mirrors the image horizontally.

Connection Type	Connection Name	Function
Input	input	The image sequence that you want to orient.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view.

Control (UI)	Default Value	Function
		For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
ImageOrient parameters continued		
orientation	No Change	Select how to rotate the input image: • **No Change** - Do not rotate the image. • **Rotate 90** - Rotate the image 90 degrees clockwise. • **Rotate 180** - Rotate the image 180 degrees clockwise. • **Rotate 270** - Rotate the image 270 degrees clockwise. • **Flip** - Mirror the image vertically (turning the image upside down). • **Flop** - Mirror the image horizontally. • **FlipFlop** - Mirror the image vertically and horizontally. This is the same as **Rotate 180**.

ImagePosition

This node applies an integer, non-resampled offset to the input image.

If you are looking to do a transform with sub-pixel re-sampling, see ImageTransform2D instead.

Connection Type	Connection Name	Function
Input	input	The image sequence you want to offset.
	out_mask	An optional image to use as a mask. By default, the offset is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view.

Control (UI)	Default Value	Function
		For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
offset		
x	0	The number of pixels by which you want to offset the input image along the x axis. For example, if you enter 2 in this field, 2 is added to the x values.
y	0	The number of pixels by which you want to offset the input image along the y axis. For example, if you enter 2 in this field, 2 is added to the y values.
adjustDisplayWindow	disabled	When enabled, the displayWindow is repositioned along with the image content. **Note:** This is very rarely desired, as convention dictates that the displayWindow should always have the lower-left corner pinned to 0, 0.

ImageReformat

Reformat lets you resize your image sequence width and height using the incoming displayWindow to determine the scale factor. This also allows you to use plates of varying image resolution on a single recipe without running into issues when combining them.

Note: If no resize is needed, filtering is NOT applied (unlike in the ImageTransform2D node, which always applies filtering).

Connection Type	Connection Name	Function
Input	bg	The background image.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the Common 2D Nodes Widget Type in Common Parameter Widgets.
resolution		
[resolution]	Dependent on Project Settings	The format to which you want to output the image sequence. The default setting resizes the image to the format indicated in the Project Settings.
width	globals.width	Sets a custom reformat width. **Note:** You can only edit this field when the initial resolution is modified.
height	globals.height	Sets a custom reformat height. **Note:** You can only edit this field when the initial resolution is modified.
preserveAspect	enabled	When enabled, Katana preserves the input image's aspect ratio.
allowOverscan	disabled	This allows the node to generate overscan (if possible). Overscan refers to image pixel data outside of the displayWindow and can be inspected using options in the Monitor.
preserveAspect: enabled		
center	enabled	When enabled, Katana pads the output image if any gaps remain after reformatting while preserving the original aspect ratio.

Control (UI)	Default Value	Function
filtering		
downFilter	Lanczos3	The filter kernel to use for downsampling: • **Gaussian** • **Triangle** • **Box** • **Bell** • **BSpline** • **Sinc** • **Lanczos2** • **Lanczos3** • **Lanczos5** • **Mitchell** • **Bilinear** • **Bicubic** • **Nearest**
upFilter	Mitchell	The filter kernel to use for upsampling: • **Gaussian** • **Triangle** • **Box** • **Bell** • **BSpline** • **Sinc** • **Lanczos2** • **Lanczos3** • **Lanczos5** • **Mitchell** • **Bilinear** • **Bicubic** • **Nearest**
highlightCompensation	enabled	When enabled, Katana adaptively compresses pixel

Control (UI)	Default Value	Function
		values prior to transform filtering and re-expands them afterward. This helps to reduce the ringing in high-contrast areas that can be a problem in linear floating point images.
clampOutput	enabled	Filtering can introduce negative values and send values above 1.0. When **clampOutput** is enabled, Katana clamps the RGB channels low at 0 and the alpha channel between 1 and 0 after the image is filtered. This is recommended for transforms on color/alpha images. When **clampOutput** is disabled, no clamping is done and values below 0 and above 1 are allowed. This is recommended for transforms applied to images that contain data that may (correctly) range more widely.

ImageTransform2D

ImageTransform2D lets you not only translate elements, but also rotate, scale, and shear them.

Connection Type	Connection Name	Function
Input	input	The image sequence you want to transform.
	out_mask	An optional image to use as a mask. By default, the merge is limited to the non-black areas of the mask.

Control (UI)	Default Value	Function
[2D node controls]	N/A	Set the controls for the stereo view. For more information, refer to the

		Common 2D Nodes Widget Type in Common Parameter Widgets.
transform		
order	trsx	Sets the operation order for translate (t), rotate (r), scale (s), and shear (x).
translate x, y	0, 0	Translates the image along the x and y axes.
rotate	0	Rotates the image around the **pivot x y** coordinates.
aspectRatio	1	Sets the pixel aspect ratio. This allows you to maintain aspect ratio when rotating anamorphic images.
scale x, y	1, 1	Scales the image width and height around the **pivot x y** coordinates.
shear x, y	0, 0	Shears the image around the **pivot x y** coordinates.
pivot x, y	(getDisplayWindow().x1+ getDisplayWindow().x0)/2, (getDisplayWindow().y1+ getDisplayWindow().y0)/2	Sets the center of rotation, scale, and shear on the x and y axes.
invert	disabled	When enabled, any transform you applied using the **translate**, **rotate**, **scale**, **shear**, or **pivot** controls is inverted.
filtering		
downFilter	Lanczos3	The filter kernel to use for downsampling: • **Gaussian** • **Triangle** • **Box** • **Bell** • **BSpline**

		• **Sinc** • **Lanczos2** • **Lanczos3** • **Lanczos5** • **Mitchell** - remapped pixels receive some smoothing, plus blurring to hide pixelation. • **Bilinear** - gives good results, but can produce square artifacts at extreme zoom. • **Bicubic** - provides more rounded results, slightly blurrier but without the square artifacts. • **Nearest** - preserves edge detail, but gives quite "blocky" textures.
upFilter	Lanczos3	The filter kernel to use for upsampling: • **Gaussian** • **Triangle** • **Box** • **Bell** • **BSpline** • **Sinc** • **Lanczos2** • **Lanczos3** • **Lanczos5** • **Mitchell** - remapped pixels receive some smoothing, plus blurring to hide pixelation. • **Bilinear** - gives good results, but can produce square artifacts at extreme zoom. • **Bicubic** - provides more rounded results, slightly blurrier but without the square artifacts.

		• **Nearest** - preserves edge detail, but gives quite "blocky" textures.
highlightCompensation	enabled	When enabled, Katana adaptively compresses pixel values prior to transform filtering and re-expands them afterward. This helps to reduce the ringing in high-contrast areas that can be a problem in linear floating point images (as we have in Katana).
clampOutput	enabled	Filtering can introduce negative values and send values above 1.0. When **clampOutput** is enabled, Katana clamps the RGB channels low at 0 and the alpha channel between 1 and 0 after the image is filtered. This is recommended for transforms on color/alpha images. When **clampOutput** is disabled, no clamping is done and values below 0 and above 1 are allowed. This is recommended for transforms applied to images that contain data that may (correctly) range more widely.
onlyApplyMotion	disabled	When enabled, Katana does not apply the node's full transform. Instead, it only applies the motion-vector component of the transform to the incoming image. If you apply **onlyApplyMotion** to the incoming image, and then transform the result by the node (with motion blur disabled), the results are similar (except for sampling differences).
motionBlur		
enable	globals.compDefaults.	When enabled, you can add motion blur

	motionBlur.enable	to the transform.
linearParamSubframeInterp	enabled	When enabled, use a fast sampling of the parameters using slerped end points for each sub-frame of motion blur. This is preferable in all cases except where lengthy blur strokes undergo **subframe acceleration**.
shutter	globals.compDefaults. motionBlur.shutter.i0, globals.compDefaults. motionBlur.shutter.i1	Sets the open and close time of the shutter when motion blurring, relative to the current frame. Changing the second number is the primary way to control the amount of motion blur applied. For example, a value of 0.5 corresponds to half a frame. Increasing the value produces more blur, and decreasing the value less.
numSamples	globals.compDefaults. motionBlur.numSamples	Sets the number of motion blur samples to compute and merge. Increase the value to produce more samples for higher quality, or decrease it to shorten the processing time. The higher the value, the smoother the result.

3D Nodes

The nodes in this section are 3D nodes that you can use within Katana. These are listed alphabetically, and each node includes a short description followed by a list of the node's parameters and their functions.

Constraint Nodes

The following section describes Katana's 3D **Constraint** nodes.

AimConstraint

Applies an aim constraint to an object in the scene graph.



Note: Aim constraints are undefined if **baseAimAxis** and **baseUpAxis** are collinear.



Note: Applying an aim constraint to a location that is already correctly oriented is likely to result in an arbitrary rotation about the aim axis.

Connection Type	Connection Name	Function
Input	input	The object to which you want to apply the aim constraint.

Control (UI)	Default Value	Function
basePath	None	Describes the scene graph location of the object to constrain. The **basePath** parameter options are available

Control (UI)	Default Value	Function
		in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
targetPath	None	Describes the object(s) location to which the **basePath** object is constrained. The **targetPath** parameter options are available by clicking **Add Locations or** ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
targetOrigin	Object	Sets how the center of the target object is calculated: • **Object** - uses the local origin of the object as the target. • **Bounding Box** - uses the center of the object's bounding box as the target. • **Face Center Average** - uses the face center average of the object as the target. • **Face Bounding Box** - uses the face center average of the object's bounding box as the target.
baseAimAxis	0.0, 0.0, -1.0	The axis of the base object that is pointed at the target. Adjusting these values changes the axis of the object that is aimed at the target.
baseUpAxis	0.0, 1.0, 0.0	The axis of the base object that is pointed upwards relative to the target. Adjusting these values changes the rotation of the base object, while keeping the aim constant.
targetUpAxis	0.0, 1.0, 0.0	The world space axis from the target object's position that defines the up direction for the base object Adjusting these values changes the axis of the base

Control (UI)	Default Value	Function
		object's up axis.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

BillboardConstraint

Applies an aim constraint to an object in a scene. To get the best possible aim, the constraint only rotates around the axis defined by **baseRotateAxis**.

Connection Type	Connection Name	Function
Input	input	The object to which you want to apply the constraint.

Control (UI)	Default Value	Function
basePath	N/A	Describes the scene graph location of the object to constrain. The **basePath** parameter options are available

Control (UI)	Default Value	Function
		in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
targetPath	N/A	Describes the object(s) location to which the **basePath** object is constrained. The **targetPath** parameter options are available by clicking **Add Locations or** ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
targetOrigin	object	Sets how the center of the target object is calculated: • **object** - uses the local origin of the object as the target. • **boundingBox** - uses the center of the object's bounding box as the target.
baseAimAxis	0.0, 0.0, -1.0	The axis of the base object that is pointed at the target. Adjusting these values changes the axis of the object that is aimed at the target.
baseRotateAxis	0.0, 1.0, 0.0	The axis of the base object that is rotated to maintain orientation to the target. Adjusting these values changes the rotation of the base object, while keeping the aim constant.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint

Control (UI)	Default Value	Function
		depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

CameraScreenWindowConstraint

This node is used to orient, scale, and position the base scene graph location so that it sits at a specified distance from the camera and fits the camera screen window exactly. This is mainly useful for registering primitive plane objects with the camera for rendering through the 2D ImagePlane and MultiPlane nodes, but there are also other creative applications.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the base scene graph location to the camera screen window.

Control (UI)	Default Value	Function
basePath	N/A	Describes the scene graph location of the object to constrain. This should be plane geometry. The **basePath** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.

Control (UI)	Default Value	Function
targetPath	N/A	Describes the camera location to which the **basePath** object is constrained. The **targetPath** parameter options are available in either a scene graph widget or dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
distance	1	Sets distance from the camera at which the base scene graph location is constrained.
planeType	XY	The type of plane that is constrained: • **XY** • **XZ**
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

ClippingConstraint

This node adjusts the camera's near and far clipping planes to fit just in front of and behind the target (along the axis from the camera). You can view the results of the ClippingConstraint node by turning on Scenegraph Implicit Resolvers at the top.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the camera's clipping planes.

Control (UI)	Default Value	Function
basePath	N/A	Describes the scene graph location of the object to constrain. The **basePath** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
targetPath	N/A	Describes the object(s) location to which the **basePath** object is constrained. The **targetPath** parameter options are available by clicking **Add Locations or** ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
pad		
near	0	Sets the amount of space to leave between the nearest extent of the target and the camera's near clipping plane.
far	0	Sets the amount of space to leave between the farthest extent of the target and the camera's far clipping plane.
ClippingConstraint parameters continued		
respectMotionBlur	Yes	When set to **Yes**, constraints are adjusted to allow for the target's motion within the time the shutter is open.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated.

Control (UI)	Default Value	Function
		The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

DollyConstraint

This node translates the camera along its look at (or local Z) axis, moving it towards or away from the target. DollyConstraint ensures that the target fits exactly in the camera's screen window and is useful for turntable setup.

See also FOVConstraint.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to apply the camera translation.

Control (UI)	Default Value	Function
basePath	N/A	Describes the scene graph location of the camera or light to dolly. The **basePath** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
targetPath	N/A	Sets the location of the object(s) to fit within the field of view. The **targetPath** parameter options are available by

Control (UI)	Default Value	Function
		clicking **Add Locations or** ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
targetBounds	box	The type of bounds to use for the target object(s): • **box** - the camera is constrained to fit the bounding box of the target object(s). • **sphere** - the camera is constrained to fit a sphere that encloses the bounding box of the target objects(s).
angleOffset	0	Sets the angle to add to the FOV in the dolly calculation.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the **Scene Graph**. When set to **No**, produce an error on constraint resolution if the target is missing.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

FOVConstraint

This node constrains the field of view of a camera to fit the target geometry. FOVConstraint closes or opens the field of view of a camera from all sides while the center of the frame remains the same. If an object is located at the edge of a light's view, the FOVConstraint should be combined with an AimConstraint to tighten the view right on the object.

See also DollyConstraint.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the field of view of a camera.

Control (UI)	Default Value	Function
basePath	N/A	Describes the scene graph location of the object to constrain. The **basePath** parameter options are available in either the scene graph widget or ▾ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
targetPath	N/A	Describes the object(s) location to which the **basePath** object is constrained. The **targetPath** parameter options are available by clicking **Add Locations or** ▾ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
targetBounds	box	Sets the type of bounds to use for the target object(s). • **box** - the field of view is constrained to fit the bounding box of the target object(s). This can be very useful for shadow maps, as it produces a tight fitting bounding box. • **sphere** - the field of view is constrained to fit a sphere that encloses the bounding box of the target object(s). This can be very useful for turntables when you don't want the field of view to change as the object rotates.
angleOffset	0	Sets the angle added to the FOV during calculation.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target

Control (UI)	Default Value	Function
		is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

OrientConstraint

OrientConstraint matches the rotation (orientation) of the object in **basePath** to the object in **targetPath**. See also ParentChildConstraint and PointConstraint.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the rotation (orientation) of an object.

Control (UI)	Default Value	Function
basePath	N/A	Sets the location of the object to constrain. The **basePath** parameter options are available by clicking the ▼ dropdown menu.

Control (UI)	Default Value	Function
		For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
targetPath	N/A	Sets the location of the object(s) to constrain the object in **basePath** to. The **targetPath** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
targetOrientation	Object	Sets the type of bounds to use for the target object(s): • **Object** • **Face**
When targetOrientation: Face		
targetFaceIndex	0	This is the faceset index to use as the target. You can orient to match the object itself, or a particular face of it, for example, selecting a poly by index of geometry.poly.startIndex.
OrientConstraint parameters continued		
xAxis	Enabled	Constrains the x Axis.
yAxis	Enabled	Constrains the y Axis.
zAxis	Enabled	Constrains the z Axis.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.

Control (UI)	Default Value	Function
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

ParentChildConstraint

Constrains the translate, rotate, and scale values of one object (the parent) to another (the child). See also OrientConstraint and PointConstraint.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the translation, rotation, and scale values of a parent object to those of the child.

Control (UI)	Default Value	Function
basePath	None	Defines the child object. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
targetPath	None	Defines the parent object. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint

Control (UI)	Default Value	Function
		depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

PointConstraint

Applies a constraint that translates the base object to a point defined by the target object(s). See also OrientConstraint and ParentChildConstraint.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the translation of the base object to that of the target object.

Control (UI)	Default Value	Function
basePath	None	Defines the location of the object to constrain. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
targetPath	None	Defines the location of the object(s) to constrain the object in **basePath** to. If you set multiple targets, then the constraint moves to the average center of the objects. The

Control (UI)	Default Value	Function
		parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
baseOrigin	Object	Sets how the center of the base object is calculated: • **Object** - uses the local origin of the object as the position of the base object. • **Bounding Box** - uses the center of the object's bounding box as the position of the base object.
targetOrigin	Object	Sets how the center of the target object is calculated: • **Object** - uses the local origin of the object as the target. • **Bounding Box** - uses the center of the object's bounding box as the target. • **Face Center Average** - uses the face center average of the object as the target. • **Face Bounding Box** - uses the face center average of the object's bounding box as the target.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
xAxis	disabled	Constrains the x Axis.
yAxis	disabled	Constrains the y Axis.
zAxis	disabled	Constrains the z Axis.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of

Control (UI)	Default Value	Function
		evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

ReflectionConstraint

ReflectionConstraint transforms the base object to a mirrored position opposite the target plane object.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to transform the base object.

Control (UI)	Default Value	Function
basePath	N/A	Sets the object to constrain. The **materialAssign** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
targetPath	N/A	Sets the object(s) to which the object in **basePath** is constrained. The **materialAssign** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
targetFaceIndex	0	The face number that forms the basis for the reflection

Control (UI)	Default Value	Function
		transform
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

ScaleConstraint

This node constrains the base object to the scale of the target object.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the scale of the base object to the target object.

Control (UI)	Default Value	Function
basePath	None	Sets the object to constrain. The **basePath** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Path Selection Widget Types in Common Parameter Widgets.
targetPath	None	Sets the object(s) to which the object in **basePath** is constrained. The targetPath parameter options are

Control (UI)	Default Value	Function
		dropdown menu. For more information, refer to the Path Selection Widget Types in Common Parameter Widgets.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

ScreenCoordinateConstraint

ScreenCoordinateConstraint modifies the camera screen window to fit the target object(s).

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to constrain the camera screen window to fit the target object.

Control (UI)	Default Value	Function
basePath	None	Sets the object to constrain. The **basePath** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.

Control (UI)	Default Value	Function
targetPath	None	Sets the object(s) to which the object in **basePath** is constrained. The **targetPath** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
respectMotionBlur	Yes	When set to **Yes**, constraints are adjusted to allow for the target's motion within the time the shutter is open.
keepAspectRatio	Yes	When set to **Yes**, maintain the aspect ratio of the screen window. When set to **No**, modify the aspect ratio of the screen window to fill as much of the frame as possible.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
addToConstraintList	No	Adds base path to **globals.constraintList** at **/root/world**. This is only needed for cases in which one constraint depends on another constraint already being evaluated. The **globals.constraintList** is used to specify the order of evaluation of constraints.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

Input Nodes

The following section describes Katana's 3D **Input** nodes.

AttributeFile_In

This node reads in an attribute file from a specified location and applies the attribute changes to the scene graph locations specified by the **CEL** statement.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to read in an attribute file.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
File Path	N/A	Describes the filepath to an Attributes File. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
Custom File Parser	N/A	Specifies the **.so** file with the Attributes File parser. Leave it empty to use the default one. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
Attribute Group Name	attributeFile	Specifies the name of the group attribute where the attributes from the file are stored. When empty, the attributes are stored directly under the location (without a group attribute).

Control (UI)	Default Value	Function
Apply When	immediate	Determines when the script runs: • **immediate** - the filter runs at the locations specified by the CEL statement as they are evaluated at this node's point in the graph. • **deferred** or **during katana standard resolve** - the script and its arguments are added as attributes under the scenegraphLocationModifers group attribute. When deferred, they are run later by the implicit ScenegraphLocationModifierResolve filter added at render time. When during katana standard resolve, they are evaluated by a LookFileResolve node or by the first implicit resolver if no LookFileResolve node is present. (They may be tested either by enabling implicit scene graph resolvers in the **Scene Graph** tab or with a ScenegraphLocationModifierResolve node.) • **during material resolve** - the script and its arguments are added as attributes under the material. Scene GraphLocationModifers group attribute. This is primarily intended for material scene graph locations. The material resolve process evaluates the script at the locations at which the material is assign or applied. This can be useful for building randomization or procedural control over shader parameters at the material level without having to apply materialOverride attributes at the geometry level.
Recursive Enable	No	When applying in a non-immediate state, enabling this results in the script running at every location beneath the assigned locations. In general this is more efficient than using an equivalent recursive CEL statement.
timing		
mode	Current Frame	Sets the timing mode to apply to the asset: • **Current Frame** - uses the current frame to access the attribute file. • **Hold Frame** - uses the frame specified by **holdTime** to access the attribute file.

Control (UI)	Default Value	Function
		• **Clamp Range** - forces Katana to only apply the attributes stored in the attributes file between the **inTime** and **outTime** frames. For before and after the clamp range, the frames specified by **inTime** and **outTime** respectively are used.
timing > mode: Hold Frame		
holdTime	globals.inTime (an expression)	Specifies which frame to use.
timing > mode: Clamp Range		
inTime	globals.inTime (an expression)	Specifies the start frame for the clamp range. It is also used for all frames before the **inTime**.
outTime	globals.outTime (an expression)	Specifies the end frame for the clamp range. It is also used for all frames after the **outTime**.

Lookfile Nodes

The following section describes Katana's 3D **Lookfile** nodes.

LookFileBake

Bakes a Look File for a scene graph location(s) specified in the **rootLocations** field.

Connection Type	Connection Name	Function
Input	orig	The original, or pre-existing scene graph to which the **default** or additional inputs are compared.
	default	A second scene graph that has a different pass to compare to the default. The Look File saves the difference between the original and default pass, or any additional passes.

Control (UI)	Default Value	Function
rootLocations		
rootLocations	/root/world	Sets the scene graph location(s) to bake the Look File information for. Any location under **/root/world** can be used, but it is recommended that components or assembly locations are specified. The **rootlocations** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
passes		
passes	None	Passes are typically render passes, but could also be auxiliary baking passes for generating point clouds or brickmaps. A Look File can have one or multiple passes. To add a pass, select **Add** > **Add Pass Input.** A new pass input is created on the node, and a pass name field is added to the pass list. To change the pass name, simply change the name text field supplied. **Note:** All pass names must be unique.
LookFileBake parameters continued		
saveTo	None	Sets where to store the baked Look File. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.

Control (UI)	Default Value	Function
options		
outputFormat	as directory	Specifies whether to bake the Look File to a single archive (currently Zip), or instead to a per-pass directory representation. The **directory** option is higher performance and recommended for heavy-weight use cases.
includeGlobalAttributes	No	When set to **Yes**, attributes on **/root** are stored in the baked Look File.
alwaysIncludeSelectedMaterialTrees	No	When set to **Yes**, include all material locations at or below the paths specified by selectedMaterialTreeRootLocations without regard to whether they are assigned to geometry within the scope of the rootLocations paths.
LookFileBake parameters continued		
Write Look File	N/A	Click to bake the Look File.

LookFileLightAndConstraintActivator

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 LookFile 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. Because it reads its input from a LookFile-resolved scene, you should place it after either a LookFileManager or LookFileResolve node.

Choose **Search Entire Incoming Scene...** or **Search Incoming Scene From Scene Graph Selection...** from the **Action** menu to find available lights and constraints.

- Entries are organized by asset, location, and then light and constraint paths.
- Gray entries are pending -- found by the searching tools but not yet enabled in the scene.
- Pending entries are not saved from session to session.

- Locations (entries immediately below the asset entries) may be refreshed individually by choosing **Search From Selected Locations** from the right-click menu. This option is only available when one or more location entries are selected.

To enable a pending entry, choose **Enable** from the right-click menu at any point within the hierarchy.

Enable and disable operations executed in this manner always act upon the selected entries and all of their children. Individual light and constraint paths may also be enabled by clicking on the checkbox next their names.

To enable everything at once:

1. First, choose **Search Entire Incoming Scene...** from the **Action** menu.
2. When that has completed, choose **Select All** from the **Action** menu right-click on any entry and choose **Enable**.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to activate LookFile lights and constraints by updating those lists.

Control (UI)	Default Value	Function
Action	N/A	Searches the scene graph for lights and constraints brought in by Look Files then enables or disables the results as required.

LookFileManager

LookFileManager decodes incoming Look Files that have been set up in another scene. Each Look File piece of imported geometry passed into this node must be assigned through a LookFileAssign node. Once the Look File is assigned, LookFileManager decodes the Look File into the passes set up by the look development artist using a LookFileBake node.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to decode incoming look files from other scenes.

Control (UI)	Default Value	Function
Look Files	N/A	Lists the Look Files that are being edited by the LookFileManager.
Passes	default	Lists any passes associated with the Look Files.
Add Override	N/A	Allows you to add overrides to selected Look Files.

LookFileMaterialsIn

This node loads materials from a Look File into the local scene to allow additional edits before they are applied to the scene.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to load a look file's materials.

Control (UI)	Default Value	Function
lookfile	N/A	Sets the Look File path and name. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
passName	N/A	Sets the pass name to use from the Look File.

Control (UI)	Default Value	Function
		For more information, refer to the Look File Widget Type in the Common Parameter Widgets.
asReference	No	When set to **Yes**, the material is loaded as a reference. Reading the material by reference causes any materials assigned to keep a reference to the Katana Look File from which they got their material.
locationForMaterials	Load at original location	Sets 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**.
locationForMaterials: Load at specified location		
userLocation	N/A	Specify the location that acts as a namespace for the material palette.

LookFileMaterialsOut

Use this node to write incoming materials into a Look File. This is useful for creating a material library that can be read into other scenes.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to write incoming materials to a look file.

Control (UI)	Default Value	Function
saveTo	N/A	Sets the location of the Look File to contain the material. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
options		
outputFormat	as archive	Specifies whether to write the Look File to a single archive (currently Zip), or instead to a per-pass directory representation. The **directory** option is higher performance and recommended for heavy-weight use cases.
Write Look File	N/A	Click to write the material to the specified Look File.

LookFileMultiBake

LookFileMultiBake is a convenient SuperTool that wraps multiple LookFileBake nodes into a single node. Passes are shared between nodes, and the Look Files can be baked individually or all at once.

Connection Type	Connection Name	Function
Input	orig	The original, or pre-existing scene graph to which the **default** or additional inputs are compared.
	default	A second scene graph or LookFileBake node that has different passes to compare to the default. The Look File saves the difference between the original and default pass, or any additional passes.

Control (UI)	Default Value	Function
	N/A	Creates a new LookFileBake and adds it to

Control (UI)	Default Value	Function
		the LookFileBake list.
When an entry has been added		
rootLocations	/root/world	Sets one or more scene graph path(s) to the location(s) to be created. The **locations** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
🔧	N/A	Brings up a searchable list to aid in selection.
saveTo	N/A	Sets where to store the baked Look File. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
Passes		
name	pass	Passes are typically render passes, but could also be auxiliary baking passes for generating point clouds or brickmaps. A LookFile can have one or multiple passes. To add a pass, select **Add** > **Add Pass Input**. A new pass input is created on the node, and a pass name field is added to the pass list. To change the pass name, simply change the name text field supplied. **Note:** All pass names must be unique.

Control (UI)	Default Value	Function
LookFileMultiBake parameters continued		
includeGlobalAttributes	No	When set to **Yes**, attributes on **/root** are stored in the baked Look File.
alwaysIncludeSelectedMaterial Trees	No	If enabled, this includes all material locations at or below the paths specified by selectedMaterialTreeRootLocations without regard to whether they are assigned to geometry within the scope of the rootLocations paths.
When alwaysIncludeSelectedMaterialTrees: yes		
selectedMaterialTreeRootLocations	/root/materials/geo	Sets one or more scene graph path(s) to the location(s) to be created. The **locations** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
Write ▼		
Write All Look Files...	N/A	Click to bake all the Look Files.
Writes Selected Look Files...	N/A	Click to bake the selected Look Files.

Control (UI)	Function
LookFileBake list [Right-click menu]	
Delete Selected Entries	Deletes the selected entries.

LookFileOverrideEnable

LookFileOverrideEnable allows you to bring into the scene materials from a specific Look File/pass so you can override them before the Look File is resolved using the LookFileResolve node.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to bring into set up an override on a look file or pass before it's resolved.

Control (UI)	Default Value	Function
lookfile	N/A	Selects the Look File to import materials from. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
versionsToOverride	Only this Version	Specifies on which version of the Look File the overrides are reflected.
locationForMaterials	Use asset name	Sets where to import the materials in the scene graph. • **Use asset name** - the materials are imported in a unique location based on the **lookfile** asset fields. • **User-specified** - provides a parameter, **userLocation**, for you to specify a namespace for the materials.
When locationForMaterials: User-specified		
userLocation	N/A	Specifies a namespace for the materials. For instance, a material at **/root/materials/geo/chrome** with userLocation default_pass is placed at **/root/materials/default_pass/geo/chrome**.
LookFileOverrideEnable parameters continued		
passName	default	Specifies the name of the Look File/pass you want to override. The entry 'default' displays if the field is left

Control (UI)	Default Value	Function
		empty. For more information, refer to the Look File Widget Type in the Common Parameter Widgets.
loadAsReference	No	When set to **Yes**, the material is loaded as a reference. Reading the material by reference causes any materials assigned to keep a reference to the Katana Look File from which they got their material.
enforceVersion	No	When set to **Yes**, you are enforcing this version to be resolved by making the LookFileResolve node resolve the asset in the LookFileOverrideEnable node instead of the one in the LookFileAssign node.

LookFileResolve

This node applies a specific pass from assigned Look Files to the scene.

Connection Type	Connection Name	Function
Input	A	The place in the node graph where you want a specific pass to be applied to the scene.

Control (UI)	Default Value	Function
passName	N/A	Sets the name of the Look File pass to use. If no look file pass is specified when attempting to resolve, the pass falls back to the default pass when **KATANA_LOOKFILE_DEFAULT_PASS_FALLBACK** is set to **1**. In this case, no error is generated and the default pass is used, otherwise Katana produces an error location.

Control (UI)	Default Value	Function
		Note: The **lookfile.resolvedPass** attribute always reports the requested pass and is, therefore, not affected by this fall-back. For more information, refer to the Look File Widget Type in the Common Parameter Widgets.
Flush Caches	N/A	Click to flush the Look File cache and force a reload.

UsdMaterialBake

The UsdMaterialBake node is a SuperTool that can be used to export material assignments, variants, shaders, and lights. Exported USD data can then be imported back into Katana or other Digital Content Creation (DCC) packages that support USD. The UsdMaterialBake node works by looking for the differences between the node's **orig** and the **default** inputs followed by each of the variant inputs in turn, if they exist. Anything differences found between **orig** and **default** or **orig** and **variant** can then be baked to a USD file and exported out of Katana. Typically, the node's origin input is connected to a scenegraph location containing the information you want to bake out.

For more information on creating locations, see LocationCreate.

Note: For versions of Katana before 4.5v1, you must to enable the USD plug-in using environment variables. For more information on how to do this, see Loading USD Plug-ins into Katana.

Note: UsdMaterialBake nodes can only be used with materials and material assignments created through native Katana nodes, they cannot be used with materials created natively with USD. To bake native USD materials, use the UsdLayerExport node. See Native USD Layer Export.

Connection Type	Connection Name	Function
Input	orig	The original, or pre-existing scenegraph to which the **default** or additional inputs are compared.

Connection Type	Connection Name	Function
	default	A second scenegraph that has a different pass to compare to the default. UsdMaterialBake saves the difference between the original and default pass, or any additional passes.

Control (UI)	Default Value	Function
rootLocations		
rootLocations	/root/world	Sets the scenegraph location(s) to bake the USD information for. Any location under **/root/world** can be used, but it is recommended that components or assembly locations are specified. The **rootlocations** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Scene Graph Locations Widget Types in Common Parameter Widgets.
Variants		
default	default	Allows you to write variants of your asset to a single USD file. Additional variants can be added by pressing the **+** button to the right of the **Variants** parameter. When additional variants are created, a new port reflecting the name of the variant becomes available on the UsdMaterialBake node. Additional information such as Network Materials or light data can then be plugged into the node, allowing you to bake out shading or lighting variants.
saveTo	None	Specifies where the exported USD file should be

Control (UI)	Default Value	Function
		saved to. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
looksFilename	shadingVariants	Allows you to set the name of the USD file to be written.
looksFileFormat	USD	Allows you to specify what USD format to write out. The available USD formats are: • **.usd** • **.usda** • **.usdc**
alwaysCreateVariantSet	No	When enabled, the USD file creates a **variantSet** type under the specified **rootPrimName**. **variantSets** are always created if there are multiple variants to bake.
createCompleteUsdAssemblyFile	No	When enabled, you can specify a payload **.usd** file so an assembly can be written out. Assembly USD files are written out as in the **.usda** format. Assembly USD files contain your geometry with all materials and shading assignments already applied. This means that additional nodes such as NetworkMaterialEdit or MaterialAssign are not needed on import, your asset is ready out of the box.
assemblyFilename	assembly	Specifies a filename so the assembly can be written out. Assembly USD files are written out as in the **.usda** format.

Control (UI)	Default Value	Function
		Note: The **assemblyFilename** parameter is only available when **createCompleteUsdAssemblyFile** is set to Yes.
payloadFilename	N/A	Allows you to specify the location of the payload USD file containing the geometry needed to export an assembly. **Note:** The **payloadFilename** parameter is only available when **createCompleteUsdAssemblyFile** is set to Yes.
rootPrimName	/root	Allows you to specify the root prim name that your data is written under. When imported, all data written through the UsdMaterialBake is contained under that root name.
variantSetName	shadingVariants	Similarly to **looksFilename**, this parameter allows you to specify a name for variant sets written from the UsdMaterialBake.

Output Nodes

The following section describes Katana's 3D **Output** nodes.

Render

The Render node takes a scene as input and renders images. The first input on this node is the scene to render. Additional inputs are dependency connections, which are used to track dependencies between passes. Each Render node is intended to be a single invocation of RenderMan or another renderer.

The Render node is really only used to track render settings, asset names, and which previously defined output passes are to be used. To set up passes, use RenderOutputDefine. To change render settings (like the active camera) use RenderSettings. To change RenderMan global settings, use PrmanGlobalSettings.

Input Information:

- Don't delete the port 'input' on the render node, or the node becomes unusable.
- Additional inputs to the Render node are dependency inputs and are only used when generating outline files for rendering.

Connection Type	Connection Name	Function
Input	input Add numbered input ports (i1, i2, i3) by pressing ▼ in the node.	The scene that you want to render out as an image.

Control (UI)	Default Value	Function
passName	Render	Sets the passName to identify this render node and is used to build the name of assets written from the Render node. When the passName is changed, the name of the Render node is also updated to stay in sync with the pass name. This is a parameter rather than just using the node name itself so you can have more control over this; node names must be unique within a Node Graph, while passName can be duplicated among different Render nodes if you need to for some reason.
lock	disabled	When enabled, the asset information for this Render node is no longer updated. This is useful when you're sharing a Render node

Control (UI)	Default Value	Function
		between shots and want to use expressions to reference the original output of the Render node. A locked Render node cannot be used to HotRender because the asset it produces is locked. It can be referenced in expressions with 'getRenderLocation'.
outputs		
outputs	N/A	Manages which available outputs are active.
farmGlobalSettings		
setActiveFrameRange	disabled	When enabled, activeFrameRange parameters are exposed to define the active frame range for rendering. When disabled, the active frame range is assumed to be the same as globals.inTime and globals.outTime.
setActiveFrameRange: enabled		
activeFrameRange > start	1	When **setActiveFrameRange** is enabled, sets the start of the active frame range.
activeFrameRange > end	1	When **setActiveFrameRange** is enabled, sets the end of the active frame range.
Render parameters continued		
dependAll	disabled	When enabled, farm dependencies wait until all frames of this node are rendered before rendering themselves.
farmFileName	N/A	The name you want to give to the generated farm file.
renderInternalDependencies	disabled	When enabled, internal dependencies of this node (input Render nodes that don't have any external (shot tree) outputs of their own) are rendered in the

Control (UI)	Default Value	Function
		same farm process as this node.
excludeFromFarmOutputGeneration	disabled	When enabled, this node does not appear in any generated farm file (however, the node is still renderable if called directly).
forceFarmOutputGeneration	disabled	When enabled, this node always appears in a generated farm file (regardless of whether it has any valid outputs). **Note:** If **excludeFromFarmOutputGeneration** is also set, the node does not appear in the generated farm file (**excludeFromFarmOutputGeneration** overrides **forceFarmOutputGeneration**).

Procedural Nodes

The following section describes Katana's 3D **Procedural** nodes.

Alembic_In

The Alembic_In node enables you to import Alembic assets. Alembic is an open source scene information interchange framework that 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. You can export to Alembic from most popular 3D applications.

Alembic caches are retrieved with reference to time, not a particular frame; because of this, Katana needs to know what frame rate to use when querying the alembic file.

Control (UI)	Default Value	Function
name	/root/world/geo/asset	Specifies the scene graph location where the Alembic asset is to be placed. The **name** parameter options are available in either the scene graph widget or ▾ dropdown menu to the right of the parameter. For more information, refer to the Create Scene Graph Location Widget Type in the Common Parameter Widgets.
abcAsset	N/A	Specifies where to retrieve the asset, an Alembic (**.abc**) file. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
addForceExpand	Yes	When set to a particular location in the scene graph, it forces expansion of the hierarchy under that location rather than stopping when a bounding box is reached.
addBounds	Root	Specifies where the overall bounds should be placed: the root location, its direct children, both, or none of these. Adding the bounds to the root has the disadvantage of producing the wrong bounds when the same root location is used by several Alembic_In nodes (the last one loaded overwrites the bounds added by the other ones). Adding the bounds to the direct children of the root location has the disadvantage of repeating the same overall bounds on each child, which means that these bounds can be potentially bigger than the real bounds of that child.

Control (UI)	Default Value	Function
fps	24	Sets how many frames constitute a second inside the Alembic file.
addToCameraList	No	For archives expected to contain cameras, this enables a light-weight traversal of the archive at **/root/world** so that the camera paths may be included in globals.cameraList. This parameter is disabled by default as it does work at a shallower point in the scene, independent of any downstream actions. In typical cases, the initial traversal is trivial and is then cached. Even so it's good practice to enable this only when necessary.
timing		
mode	Current Frame	Sets the timing mode to apply to the asset: • **Current Frame** - uses the current frame to access the Alembic asset. • **Hold Frame** - uses the frame specified by **holdTime** to access the Alembic asset. • **Clamp Range** - forces Katana to only retrieve geometry from between the **inTime** and **outTime** frames. The frames specified by **inTime** and **outTime** are used for frames before and after the clamp range respectively.
timing > mode: Hold Frame		
holdTime	globals.inTime (an expression)	The frame to retrieve from the Alembic asset.
timing > mode: Clamp Range		
inTime	globals.inTime (an expression)	The start frame for retrieving geometry from the Alembic asset.

Control (UI)	Default Value	Function
outTime	globals.outTime (an expression)	The end frame for retrieving geometry from the Alembic asset.
advanced		
useOnlyShutterOpenCloseTimes	No	When set to **Yes**, it forces the Alembic cache to only use the time samples corresponding to shutter open and close times when '**maxTimeSamples**' is set to 2 in a RenderSettings node.

RendererProceduralArgs

The RendererProceduralArgs node allows you to use and declare renderer procedurals, such as native Arnold or RenderMan procedurals. For example you can use procedurals to generate hair or other geometry. Procedurals are assigned to scene graph locations of type 'renderer procedural' with the RendererProceduralAssign node.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to declare and use procedurals from a native renderer.

Control (UI)	Default Value	Function
name	RendererProceduralArgs	Sets the name of the procedural.
action	create new location	Determines what action the node takes: • **create new location** - creates a new scene graph location of type "renderer procedural arguments" beneath **/root/materials/proc** with the name specified by the **name** parameter. • **inherit from existing location** - creates a new scene

Control (UI)	Default Value	Function
		graph location of type "renderer procedural arguments" beneath the location specified by **inheritsFrom.location** parameter with the name specified by the **name** parameter. • **edit existing location** - displays the incoming values of a single scene graph location of type "renderer procedural arguments" or "renderer procedural" specified by the **edit.location** parameter - applies the specified edits to the values at that location. • **edit multiple locations** - edits values at the locations specified by the CEL statement from the **edit.CEL** parameter. This does not display incoming values because they could differ from location to location. This means that you must specify the dso path for the procedural in order to display settable parameters.
action	(continued)	• **define overrides** - accepts drag-and-dropped attributes from rendererProcedural attribute groups. This can be used in two ways: 1. When aimed (via CEL) at locations within the renderable scene, it creates a **rendererProceduralOverride** attribute. At resolve time, these values override equivalent values in the **rendererProcedural** attribute of scene graph locations of type "RendermanProcedural" beneath. This is useful for making global changes to many different procedurals at once, regardless of whether they share the same source. 2. When aimed (via CEL) at locations of type "renderer procedural arguments", it modifies the rendererProcedural directly in the same manner as the "edit multiple locations" action. • **remove overrides** - removes or masks inherited attributes in the **rendererProceduralOverride** at the scene graph locations specified by the CEL statement

Control (UI)	Default Value	Function
		of the overrides.CEL parameter.
When action is: create new location		
namespace	N/A	Specifies the scene graph location under **/root/materials** where to place the procedural.
procedural	N/A	Brings up the file browser or your studio's asset management browser and enables you to select the procedural to use. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
args	N/A	Specifies the args file attributes.
useInfiniteBounds	Yes	Controls whether the declaration of the procedural includes bounds that the renderer can use to defer evaluation of the procedural until it needs data from inside those bounds. When set to **Yes**, nearly infinite bounds are used. When set to **No**, no bounds are used.
includeCameraInfo	None	Specifies the format in which the camera information is passed to the procedural by the render plug-in. It includes the following: camera's transform, field of view, and screen window. • **None** - no camera information is passed to the procedural. • **As Parameters** - the camera information is passed into the procedural as arguments to the procedural. So the procedural you're specifying args for needs to be expecting to receive this camera information, and know what to do with it. • **As Attributes** - the camera information is put into the RIB stream as a "user" attribute. So the procedural needs to know to look for this attribute if it wants to use it.

Control (UI)	Default Value	Function
		The **Parameters** and **Attributes** formats are the following: • string **cameraInfo_path** - path to the camera. • float **cameraInfo_fov** - the Field of View. • float **cameraInfo_near** - distance to the near clipping plane. • float **cameraInfo_far** - distance to the far clipping plane. • float **cameraInfo_left** - screen window left. • float **cameraInfo_right** - screen window right. • float **cameraInfo_top** - screen window top. • float **cameraInfo_bottom** - screen window bottom. • float[16] **cameraInfo_xform** - the camera transform.
When includeCameraInfo is: As Parameters or As Attributes		
cameraInfo > whichCamera	Render Camera	• **Render Camera** - selects the render camera. • **Other Camera** - lets you set the path for the camera using **cameraPath**
When action is: inherit from existing location		
InheritsFrom > location	N/A	Specifies the location to use in the **Scene Graph** tab. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
procedural	N/A	Brings up the file browser or your studio's asset management browser and enables you to select the procedural to use. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.

Control (UI)	Default Value	Function
args	N/A	Specifies the args file attributes.
useInfiniteBounds	Yes	Controls whether the declaration of the procedural includes bounds that the renderer can use to defer evaluation of the procedural until it needs data from inside those bounds. When set to **Yes**, nearly infinite bounds are used. When set to **No**, no bounds are used.
includeCameraInfo	None	Specifies the format in which the camera information is passed to the procedural by the render plug-in. It includes the following: camera's transform, field of view, and screen window. • **None** - no camera information is passed to the procedural. • **As Parameters** - the camera information is passed into the procedural as arguments to the procedural. So the procedural you're specifying args for needs to be expecting to receive this camera information, and know what to do with it. • **As Attributes** - the camera information is put into the RIB stream as a "user" attribute. So the procedural needs to know to look for this attribute if it wants to use it. The **Parameters** and **Attributes** formats are the following: • string **cameraInfo_path** - path to the camera. • float **cameraInfo_fov** - the Field of View. • float **cameraInfo_near** - distance to the near clipping plane. • float **cameraInfo_far** - distance to the far clipping plane. • float **cameraInfo_left** - screen window left. • float **cameraInfo_right** - screen window right.

Control (UI)	Default Value	Function
		• float **cameraInfo_top** - screen window top. • float **cameraInfo_bottom** - screen window bottom. • float[16] **cameraInfo_xform** - the camera transform.
When includeCameraInfo is: As Parameters or As Attributes		
cameraInfo > whichCamera	Render Camera	• **Render Camera** - selects the render camera. • **Other Camera** - lets you set the path for the camera using **cameraPath**
When action is: edit existing location		
edit > location	N/A	Specifies the location to use in the **Scene Graph** tab. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
procedural	N/A	Brings up the file browser or your studio's asset management browser and enables you to select the procedural to use. For more information, refer to Asset and File Path Widget Types in Common Parameter Widgets.
args	N/A	Specifies the args file attributes.
useInfiniteBounds	Yes	Controls whether the declaration of the procedural includes bounds that the renderer can use to defer evaluation of the procedural until it needs data from inside those bounds. When set to **Yes**, nearly infinite bounds are used. When set to **No**, no bounds are used.
includeCameraInfo	None	Specifies the format in which the camera information is passed to the procedural by the render plug-in.

Control (UI)	Default Value	Function
		It includes the following: camera's transform, field of view, and screen window. • **None** - no camera information is passed to the procedural. • **As Parameters** - the camera information is passed into the procedural as arguments to the procedural. So the procedural you're specifying args for needs to be expecting to receive this camera information, and know what to do with it. • **As Attributes** - the camera information is put into the RIB stream as a "user" attribute. So the procedural needs to know to look for this attribute if it wants to use it. The **Parameters** and **Attributes** formats are the following: • string **cameraInfo_path** - path to the camera. • float **cameraInfo_fov** - the Field of View. • float **cameraInfo_near** - distance to the near clipping plane. • float **cameraInfo_far** - distance to the far clipping plane. • float **cameraInfo_left** - screen window left. • float **cameraInfo_right** - screen window right. • float **cameraInfo_top** - screen window top. • float **cameraInfo_bottom** - screen window bottom. • float[16] **cameraInfo_xform** - the camera transform.
When includeCameraInfo is: As Parameters or As Attributes		
cameraInfo > whichCamera	Render Camera	• **Render Camera** - selects the render camera. • **Other Camera** - lets you set the path for the camera using **cameraPath**

Control (UI)	Default Value	Function
When action is: edit multiple locations		
edit > CEL	N/A	Sets the CEL specification of scene graph locations on which the assignment acts. The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
procedural	N/A	Brings up the file browser or your studio's asset management browser and enables you to select the procedural to use. For more information, refer to Asset and File Path Widget Types in Common Parameter Widgets.
args	N/A	Specifies the args file attributes.
useInfiniteBounds	Yes	Controls whether the declaration of the procedural includes bounds that the renderer can use to defer evaluation of the procedural until it needs data from inside those bounds. When set to **Yes**, nearly infinite bounds are used. When set to **No**, no bounds are used.
includeCameraInfo	None	Specifies the format in which the camera information is passed to the procedural by the render plug-in. It includes the following: camera's transform, field of view, and screen window. • **None** - no camera information is passed to the procedural.

Control (UI)	Default Value	Function
		• **As Parameters** - the camera information is passed into the procedural as arguments to the procedural. So the procedural you're specifying args for needs to be expecting to receive this camera information, and know what to do with it. • **As Attributes** - the camera information is put into the RIB stream as a "user" attribute. So the procedural needs to know to look for this attribute if it wants to use it. The **Parameters** and **Attributes** formats are the following: • string **cameraInfo_path** - path to the camera. • float **cameraInfo_fov** - the Field of View. • float **cameraInfo_near** - distance to the near clipping plane. • float **cameraInfo_far** - distance to the far clipping plane. • float **cameraInfo_left** - screen window left. • float **cameraInfo_right** - screen window right. • float **cameraInfo_top** - screen window top. • float **cameraInfo_bottom** - screen window bottom. • float[16] **cameraInfo_xform** - the camera transform.
When includeCameraInfo is: As Parameters or As Attributes		
cameraInfo > whichCamera	Render Camera	• **Render Camera** - selects the render camera. • **Other Camera** - lets you set the path for the camera using **cameraPath**
When action is: define overrides		
overrides > CEL	N/A	Sets the CEL specification of scene graph locations on which the assignment acts.

Control (UI)	Default Value	Function
		The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
attrs	Drop Attributes Here	Middle-click and drag attributes from the **Attributes** tab to this hotspot to use that attribute.
When action is: remove overrides		
overrides > CEL	N/A	Sets the CEL specification of scene graph locations on which the assignment acts. The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

Resolve Nodes

The following section describes Katana's 3D **Resolve** nodes.

ConstraintResolve

This node resolves all constraints stored on the locations referenced in globals.constraintList at **/root/world**.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want all constraints in the globals.constraintList to be resolved.

MaterialResolve

Resolves materials in the scene graph. At scene graph locations with **materialAssign** attributes, it finds the material that is referenced and copies its material attributes to the scene graph location. Results of this operation can be viewed in the **Attributes** tab. It can also be used to apply material overrides set by the **Material** node.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to resolve materials in the scene graph.

Source Nodes

The following section describes Katana's 3D **Source** nodes.

AttributeFile_In

This node reads in an attribute file from a specified location and applies the attribute changes to the scene graph locations specified by the **CEL** statement.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to read in an attribute file.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
File Path	N/A	Describes the filepath to an Attributes File. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
Custom File Parser	N/A	Specifies the **.so** file with the Attributes File parser. Leave it empty to use the default one. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
Attribute Group Name	attributeFile	Specifies the name of the group attribute where the attributes from the file are stored. When empty, the attributes are stored directly under the location (without a group attribute).
Apply When	immediate	Determines when the script runs: • **immediate** - the filter runs at the locations specified by the CEL statement as they are evaluated at this node's point in the graph. • **deferred** or **during katana standard resolve** - the script and its arguments are added as attributes under the scenegraphLocationModifers group attribute. When

Control (UI)	Default Value	Function
		deferred, they are run later by the implicit ScenegraphLocationModifierResolve filter added at render time. When during katana standard resolve, they are evaluated by a LookFileResolve node or by the first implicit resolver if no LookFileResolve node is present. (They may be tested either by enabling implicit scene graph resolvers in the **Scene Graph** tab or with a ScenegraphLocationModifierResolve node.) • **during material resolve** - the script and its arguments are added as attributes under the material. Scene GraphLocationModifers group attribute. This is primarily intended for material scene graph locations. The material resolve process evaluates the script at the locations at which the material is assign or applied. This can be useful for building randomization or procedural control over shader parameters at the material level without having to apply materialOverride attributes at the geometry level.
Recursive Enable	No	When applying in a non-immediate state, enabling this results in the script running at every location beneath the assigned locations. In general this is more efficient than using an equivalent recursive CEL statement.
timing		
mode	Current Frame	Sets the timing mode to apply to the asset: • **Current Frame** - uses the current frame to access the attribute file. • **Hold Frame** - uses the frame specified by **holdTime** to access the attribute file. • **Clamp Range** - forces Katana to only apply the attributes stored in the attributes file between the **inTime** and **outTime** frames. For before and after the clamp range, the frames specified by **inTime** and **outTime** respectively are used.
timing > mode: Hold Frame		

Control (UI)	Default Value	Function
holdTime	globals.inTime (an expression)	Specifies which frame to use.
timing > mode: Clamp Range		
inTime	globals.inTime (an expression)	Specifies the start frame for the clamp range. It is also used for all frames before the **inTime**.
outTime	globals.outTime (an expression)	Specifies the end frame for the clamp range. It is also used for all frames after the **outTime**.

CameraCreate

The CameraCreate node is used to create a scene graph containing a camera. CameraCreate does not load the camera from any file or asset but instead builds an entirely new camera from the parameters you specify on this node.



Note: LightCreate and CameraCreate are identical, except for the type of scene graph locations they create, and the population of the lightList vs. cameraList.

Tip: To lock a camera's position after it's created, set **makeInteractive** on the CameraCreate to **No**.

Control (UI)	Default Value	Function
name	/root/world/cam/camera	This is the scene graph location where the camera is created. For example, the default value of **/root/world/cam/camera** creates a camera at the location **/root/world/cam/camera**. The **name** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of

Control (UI)	Default Value	Function
		the parameter. For more information, refer to the Create Scene Graph Location Widget Type in the Common Parameter Widgets.
projection	perspective	Toggles the projection type of the camera: • **perspective** • **orthographic**
fov	70	Controls the field of view angle in degrees.
near	0.1	Sets the near clipping plane distance.
far	100000	Sets the far clipping plane distance.
screenWindow		
left	-1	This set of four number parameters controls the screen window placement on the imaging plane. They are, in order, left, right, bottom and top bounds of the screen window.
right	1	
bottom	-1	
top	1	
centerOfInterest	20	Offsets the center of interest of the camera.
orthographicWidth	30	Sets the orthographic projection width.
includeInCameraList	Yes	When enabled, the camera is visible in the camera list on the **/root/world** location, under the **Scene Graph** tab.
transform		
transform	N/A	Transforms the camera according to the SRT or matrix controls. For more information, refer to the Transform Controls Widget Type in the Common Parameter Widgets.

Control (UI)	Default Value	Function
transform > Tools	N/A	Adjusts the camera to match selected scene graph selection options in the dropdown menu. For more information, refer to the Transform Tools Widget Type in the Common Parameter Widgets.
CameraCreate parameters continued		
makeInteractive	Yes	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.

CameraImagePlaneCreate

Note: Although CameraImagePlaneCreate nodes are available in Katana versions 4.5v1 and above, the ability to set up Image Planes is no longer supported due to the OpenSceneGraph-based Viewer being replaced with the Hydra Viewer.

Creates attributes on a camera that describe an image plane. In the Viewer, the camera displays the image plane.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to create attributes on a camera for an image plane.

Control (UI)	Default Value	Function
cameraLocation	N/A	This is the scene graph location where the target camera resides. For example, the default value of **/root/world/cam/camera** references a camera at the location **/root/world/cam/camera**.

Control (UI)	Default Value	Function
		For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
name	plane	Sets the name of the image plane created.
imagePath	N/A	Sets the filepath of the image or sequence to display in the image plane. Supported file formats include **.cin**, **.dpx**, **.rla**, **.iff**, **.tif**, **.jpg**, **.tga**, **.rgb**, and **.tga**. Floating point data (**.exr**, **.tif**, **.zfile**) is not currently supported.  **Note:** Image sequences must contain a padded frame number. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
frame	1	Sets the frame of the image sequence to use.
depth	10000	Sets the distance from the camera to image plane.
alpha	1	Sets the image plane's alpha value.
displayOnlyIfCurrent	No	When set to **Yes**, this image plane is only displayed when looking through the camera it is attached to. When set to **No**, you can see the image plane in all views.
displayMode	RGBA	Sets the image plane display mode: • **None** • **Outline** • **RGB** • **RGBA**
fit	Best	Controls how the image file fits into the image plane if there is a mismatch between aspect ratios: • **Fill** - the image is scaled as required to fill the plane,

Control (UI)	Default Value	Function
		without being squashed or stretched. Any excess is cropped. • **Best** - the image is scaled as required to display it entirely within the plane, without being squashed, stretched or cropped. • **Horizontal** - the image is scaled as required so that its aspect ratio is maintained within the horizontal bounds of the plane. Any excess at the top or bottom is cropped. • **Vertical** - the image is scaled as required so that its aspect ratio is maintained within the vertical bounds of the plane. Any excess at the left or right is cropped. • **To Size** - the image is stretched or squashed to fit with the plane both horizontally and vertically. **Note:** To control the image's SRT values directly, enable the **manualPlacementSettings** checkbox.
crop		
left	0	Sets the amount of manual crop to apply to the edges of the image plane.
bottom	0	
right	1	
top	1	
CameraImagePlaneCreate parameters continued		
manualPlacementSettings	No	When set to **Yes**, you can adjust the image manually using its SRT values. Otherwise, the image plane placement is synced to the camera.
manualPlacementSettings: Yes > size		

Control (UI)	Default Value	Function
Size controls the width and height of the image plane.		
x	1	Adjusts the size of the image plane along the x axis.
y	1	Adjusts the size of the image plane along the y axis.
manualPlacementSettings: Yes > offset Offset controls how much the center of the image plane is offset from the center of the viewing frustum of the camera.		
x	0	Adjusts the offset of the image plane along the x axis.
y	0	Adjusts the offset of the image plane along the y axis.
manualPlacementSettings: Yes		
rotate	0	Adjusts the rotation of the image plane around the view vector.

CollectionCreate

Collections are used to store a CEL statement. They are stored as attributes in the location given by the **location** parameter (see below). As they are simply attributes within the scene graph, collections can be included within Katana look files.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to create a collection.

Control (UI)	Default Value	Function
location	N/A	The scene graph location where the collection is saved. The **location** parameter options are available in either the

Control (UI)	Default Value	Function
		scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
name	N/A	Sets the name of the collection.
CEL	N/A	Specifies scene graph locations to store as part of this collection. The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
stripDisabledBlocks	No	When set to **Yes**, this removes parts of a collection expression that have been disabled by pressing the **D** key over a CEL editor panel. CEL statement parts that are disabled are normally wrapped in **#IGNORE**(...) statements. When **stripDisabledBlocks** is set to **Yes**, **#IGNORE**(...) blocks are removed from the resulting collection expression. This optional optimization means extra work upfront, but results in CEL statements that are potentially much shorter.

CoordinateSystemDefine

Creates a named coordinate system accessed by PRMan shaders. The list of all global named coordinate systems can be found in **/root/world**, **globals.coordinateSystems**.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to create a named coordinate system.

Control (UI)	Default Value	Function
scope	global	Specifies how the coordinate system is defined. The options are: • **global** • **relative scope**
coordinateSystemName	N/A	Specifies the unique name of the coordinate system to create.
scope: global		
referenceLocation	N/A	Specifies the scene graph location whose global transform defines the coordinate system. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
scope: relative scoped		
baseLocation	N/A	Specifies the scene graph location whose scoped transform defines the base coordinate system. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
relativeLocation	N/A	Specifies the scene graph location whose scoped transform defines the relative coordinate system. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.

InfoCreate

This node creates a hierarchy of info locations, each tagged with the specified xml block. If **leafName** is specified, locations named with the **leafName** are created as children of the specified locations. If **leafName** is left empty, info locations are created directly at the specified locations.

Images can be embedded using standard syntax, however the node cannot reference web servers (must be links in the file system).

Extra scene graph locations can be baked into Look Files (**.klf**) and are added as new scene graph locations in the scene when a Look File is resolved. A common use of the InfoCreate node is to provide documentation and/or version specific information (either baked in a Look File or as an InfoCreate node in a macro).

Control (UI)	Default Value	Function
leafName	info	If a **leafName** is populated, the info is created below each specified item in the locations parameter array. Common leaf names are: readme, info, and user.
locations	/root/world	If **leafName** is not populated, **info** locations are created directly at the specified locations. If **leafName** is specified, locations named with the **leafName** are created as children of the specified locations. The **locations** parameter options are available by clicking **Add Locations or** ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets. **Note:** An empty or invalid location value generates a location of type **error** under **/root**.
text (html editor)	view editor and preview	Sets the mode for the html editor: • **view editor and preview** - the top section of the editor is html source and the bottom section of the editor is a rendered preview.

Control (UI)	Default Value	Function
		• **view only editor** - shows only the top section of editor (html source). • **view only preview** - shows only the bottom section of editor (rendered preview).

LightCreate

This node is used to create a scene graph containing a light. LightCreate does not load the light from any file or asset but instead builds an entirely novel light from the parameters you specify on this node. This node is not used generally, the GafferThree node is often used instead.



Note: LightCreate requires a light shader to function properly.

LightCreate and CameraCreate are identical, except for the type of scene graph locations they create, and the population of the lightList vs. cameraList.

Control (UI)	Default Value	Function
name	/root/world/lgt/light	Sets the scene graph location where the light is created. For example, the default value of **/root/world/lgt/light** creates a light at the location **/root/world/lgt/light**. The **name** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Create Scene Graph Location Widget Type in the Common Parameter Widgets.
projection	perspective	Sets the light projection mode: • **perspective** • **othographic**

Control (UI)	Default Value	Function
fov	70	Controls the field of view angle in degrees.
near	0.1	Sets the near clipping plane distance.
far	100000	Sets the far clipping plane distance.
screenWindow		
left	-1	This set of four number parameters controls the screen window placement on the imaging plane. They are, in order, left, right, bottom and top bounds of the screen window.
right	1	
bottom	-1	
top	1	
LightCreate parameters continued		
centerOfInterest	20	Offsets the center of interest of the light.
orthographicWidth	30	Sets the orthographic projection width.
radius	1	Sets the light's radius.
initialState	on	Determines whether the newly-added light location is initially on or off.
previewColor		
previewColor	1, 1, 0	Specifies the color of the light in the Viewer. This value does not affect the color value of the light when rendering, it's used for testing the placement of lights. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
manipulators > Add		
Add Entry	N/A	Adds entries to the manipulators list.
manipulators > manipulator		
name	N/A	The name of the manipulator you've created.

Control (UI)	Default Value	Function
preset	default	The preset manipulator type.
selected	Handles	When selected, how the manipulator displays: • **Handles** - the manipulator is shown with handles. • **Outline** - the manipulator is shown as an outline around the light. • **None** - the manipulator display doesn't change when selected.
visible	Outline	When visible, how the manipulator displays: • **Handles** - the manipulator is shown with handles. • **Outline** - the manipulator is shown as an outline around the light. • **None** - the manipulator display doesn't appear.
transform		
transform	N/A	Transforms the light according to the SRT or matrix controls. For more information, refer to the Transform Controls Widget Type in the Common Parameter Widgets.
transform > Tools	N/A	Adjusts the light to match selected scene graph selection options in the dropdown menu. For more information, refer to the Transform Tools Widget Type in the Common Parameter Widgets.
LightCreate parameters continued		
makeInteractive	Yes	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.

LocationCreate

Allows you to create a scene graph location of any type. Often used in macros to generate one or more scene graph location without the overhead or type-specific attributes created by the other Create nodes.

Control (UI)	Default Value	Function
type	group	Sets the type attribute of the scene graph location(s) to be created (as seen in the 'Type' column of the scene graph).
locations	/root/world	Sets one or more scene graph path(s) to the location(s) to be created. The **locations** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
attrs	None	Drag string or number attributes here to have them added to the scene graph location(s) created by this node.

Material

This node defines a material, which is a set of shader calls and associated parameters. Materials are assigned to geometry using the MaterialAssign node.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to define a material.

Control (UI)	Default Value	Function
name	Material	Sets the node name. It's a good idea to use a meaningful name such as "mtl_red".
action	create new material	Determines the node's behavior: • **create new material** - creates a new scene graph location of type **geometry material** or **light material** beneath **/root/materials/**(geo|lgt) with the name specified by the

Control (UI)	Default Value	Function
		name parameter. • **create from LookFile** - creates a new scene graph location of type **geometry material** or **light material** from a specified LookFile with the **name** specified by the name parameter. • **create child material** - creates a new scene graph location of type **geometry material** or **light material** beneath the location specified by **inheritsFrom.location** parameter with the **name** specified by the name parameter. • **edit material** - displays the incoming values of a single scene graph material location specified by the **edit.location** parameter. This is useful for making changes when the original Material node, which created this location, is not within the current session, or for multiple branches of a graph.
action	(continued)	• **override materials** - accepts drag-and-dropped attributes from material attribute groups. This can be used in two ways: •When aimed at locations within the renderable scene, it creates a materialOverride attribute. At resolve time, these values override equivalent values in the material attribute of renderable scene graph locations beneath. This is useful for making global changes to the assigned instances of many different materials at once, regardless of whether they share the same source. •When aimed at locations of type **geometry material** or **lightmaterial**, it modifies the material directly. This does not display incoming values because they could differ from location to location. This means that you must specify the shader in order to display adjustable parameters.
When action is: create new material		

Control (UI)	Default Value	Function
namespace	N/A	Specifies the scene graph location where the material is placed.
makeInteractive	No	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
Add shader	N/A	Click to add a renderer-specific shader to the material. The shaders that are available change depending on the renderers installed.
When action is: create from LookFile		
namespace	N/A	Specifies the scene graph location where the material is placed.
makeInteractive	No	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
lookfile	N/A	Selects the Look File to import materials from. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
materialPath	N/A	Allows you to select materials from the Look File.
asReference	Yes	When set to **Yes**, the material is loaded as a reference. Reading the material by reference causes any materials assigned to keep a reference to the Katana Look File from which they got their material.
When action is: create child material		
makeInteractive	No	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
Add shader	N/A	Click to add a renderer-specific shader to the material. The shaders that are available change depending on the renderers installed.
inheritsFrom > location	N/A	Sets the scene graph path to the location to be created. The **location** parameter options are available by clicking the dropdown menu.

Control (UI)	Default Value	Function
		For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
When action is: edit material		
makeInteractive	No	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
Add shader	N/A	Click to add a renderer-specific shader to the material. The shaders that are available change depending on the renderers installed.
edit > location	N/A	Sets the scene graph path to the location to be created. The **locations** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
When action is: override materials		
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
attrs	N/A	Middle-click and drag attributes from the **Attributes** tab to this hotspot to use that attribute.

PrimitiveCreate

Adds a primitive geometry element to a scene such as sphere, cube, or cylinder as well as renderer procedural, rib archive, brickmap and clipping plane.



Tip: You can directly create PrimitiveCreate nodes by pressing **P** in the Node Graph.

Control (UI)	Default Value	Function
name	/root/world/geo/primitive	Describes the scene graph location where the object is created. The **name** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Create Scene Graph Location Widget Type in the Common Parameter Widgets.
type	sphere	Sets the type of primitive created (plane or sphere, for example)
transform		
transform	N/A	Transforms the primitive according to the SRT or matrix controls. For more information, refer to the Transform Controls Widget Type in the Common Parameter Widgets.
transform > Tools ▼	N/A	Adjusts the primitive to match selected scene graph selection options in the dropdown menu. For more information, refer to the Transform Tools Widget Type in the Common Parameter Widgets.
PrimitiveCreate parameters continued		
makeInteractive	Yes	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
viewerPickable	Yes	When set to **Yes**, the object can be selected in the Viewer. When set to **No**, the object can only be selected through the scene graph.

TeapotCreate

This node creates a specific type of PrimitiveCreate node rendering a teapot instead of a sphere or cube. See also PrimitiveCreate.

Control (UI)	Default Value	Function
name	/root/world/geo/primitive	Describes the scene graph location where the object is created. The **name** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Create Scene Graph Location Widget Type in the Common Parameter Widgets.
type	teapot	Sets the type of primitive created (plane or sphere, for example)
transform		
transform	N/A	Transforms the teapot according to the SRT or matrix controls. For more information, refer to the Transform Controls Widget Type in the Common Parameter Widgets.
transform > Tools ▼	N/A	Adjusts the teapot to match selected scene graph selection options in the dropdown menu. For more information, refer to the Transform Tools Widget Type in the Common Parameter Widgets.
TeapotCreate parameters continued		
makeInteractive	Yes	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
viewerPickable	Yes	When set to **Yes**, the object can be selected in the Viewer.

Control (UI)	Default Value	Function
		When set to **No**, the object can only be selected through the scene graph.

SuperTool Nodes

The following section describes Katana's 3D **SuperTool** nodes.

Importomatic

The Importomatic is a SuperTool node with a custom interface to load and manage different geometry types. Other geometry or asset types can be added using a custom plug-in.

- Geometry or asset types may be grouped by adding additional outputs, then middle-dragging loaded geometry or assets under the new output.
- Additional outputs may be renamed by selecting the additional output in the Name column, then editing the output name in the parameter field. The default output can not be reordered or renamed.
- The order in which the geometry appears in the GUI determines the merge order, and its listing place in the scene graph.
- Multiple geometry entries may be selected at once, but their parameters are not displayed in the GUI. Multiple entries may be selected, moved, and regrouped at once.
- If a geometry asset has version information, it is displayed in the Version column. A version can be selected by left-clicking on the triangle in the version column for a geometry listing, toggling the Show Explicit Versions button, and selecting the desired version.

Right-click Menu

The right-click menu options available for each geometry asset allow a user to ignore or delete selected asset entries. Additional outputs can also be deleted from the right-click menu. The default output can not be deleted.

Levels of detail, if available for that asset type can be activated, by selecting Include Levels of Detail from the right-click menu. The Status column indicates that LODs are enabled.

LookFileLightAndConstraintActivator

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 LookFile 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. Because it reads its input from a LookFile-resolved scene, you should place it after either a LookFileManager or LookFileResolve node.

Choose **Search Entire Incoming Scene...** or **Search Incoming Scene From Scene Graph Selection...** from the **Action** menu to find available lights and constraints.

- Entries are organized by asset, location, and then light and constraint paths.
- Gray entries are pending -- found by the searching tools but not yet enabled in the scene.
- Pending entries are not saved from session to session.
- Locations (entries immediately below the asset entries) may be refreshed individually by choosing **Search From Selected Locations** from the right-click menu. This option is only available when one or more location entries are selected.

To enable a pending entry, choose **Enable** from the right-click menu at any point within the hierarchy.

Enable and disable operations executed in this manner always act upon the selected entries and all of their children. Individual light and constraint paths may also be enabled by clicking on the checkbox next their names.

To enable everything at once:

1. First, choose **Search Entire Incoming Scene...** from the **Action** menu.
2. When that has completed, choose **Select All** from the **Action** menu right-click on any entry and choose **Enable**.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to activate LookFile lights and constraints by updating those lists.

Control (UI)	Default Value	Function
Action	N/A	Searches the scene graph for lights and constraints brought in by Look Files then enables or disables the results as required.

LookFileManager

LookFileManager decodes incoming Look Files that have been set up in another scene. Each Look File piece of imported geometry passed into this node must be assigned through a LookFileAssign node. Once the Look File is assigned, LookFileManager decodes the Look File into the passes set up by the look development artist using a LookFileBake node.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to decode incoming look files from other scenes.

Control (UI)	Default Value	Function
Look Files	N/A	Lists the Look Files that are being edited by the LookFileManager.
Passes	default	Lists any passes associated with the Look Files.
Add Override	N/A	Allows you to add overrides to selected Look Files.

LookFileMultiBake

LookFileMultiBake is a convenient SuperTool that wraps multiple LookFileBake nodes into a single node. Passes are shared between nodes, and the Look Files can be baked individually or all at once.

Connection Type	Connection Name	Function
Input	orig	The original, or pre-existing scene graph to which the **default** or additional inputs are compared.
	default	A second scene graph or LookFileBake node that has different passes to compare to the default. The Look File saves the difference between the original and default pass, or any additional passes.

Control (UI)	Default Value	Function
	N/A	Creates a new LookFileBake and adds it to the LookFileBake list.
When an entry has been added		
rootLocations	/root/world	Sets one or more scene graph path(s) to the location(s) to be created. The **locations** parameter options are available by clicking **Add Locations** or dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
	N/A	Brings up a searchable list to aid in selection.
saveTo	N/A	Sets where to store the baked Look File. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
Passes		
name	pass	Passes are typically render passes, but could also be auxiliary baking passes for generating point clouds or brickmaps. A

Control (UI)	Default Value	Function
		LookFile can have one or multiple passes. To add a pass, select **Add** > **Add Pass Input**. A new pass input is created on the node, and a pass name field is added to the pass list. To change the pass name, simply change the name text field supplied. **Note:** All pass names must be unique.
LookFileMultiBake parameters continued		
includeGlobalAttributes	No	When set to **Yes**, attributes on **/root** are stored in the baked Look File.
alwaysIncludeSelectedMaterial Trees	No	If enabled, this includes all material locations at or below the paths specified by selectedMaterialTreeRootLocations without regard to whether they are assigned to geometry within the scope of the rootLocations paths.
When alwaysIncludeSelectedMaterialTrees: yes		
selectedMaterialTreeRootLocations	/root/materials/geo	Sets one or more scene graph path(s) to the location(s) to be created. The **locations** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
Write ▼		
Write All Look Files...	N/A	Click to bake all the Look Files.
Writes Selected Look Files...	N/A	Click to bake the selected Look Files.

Control (UI)	Function
LookFileBake list [Right-click menu]	
Delete Selected Entries	Deletes the selected entries.

Other 3D Nodes

The following section describes Katana's **Other** 3D nodes.

ArnoldObjectSettings

The ArnoldObjectSettings node is populated by the XML files located at **${KATANA_ROOT}/plugins/Resources/Arnold<version number>/GenericAssign**. These parameter names and defaults can change between Arnold versions and, as such, they are provided for you to change.



Note: The parameters that are available for this node are dependent on which version of Arnold you are using. As such, only renderer-agnostic parameters are listed. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with Arnold.

Connection Type	Connection Name	Function
Input	input	The 3D input to which you want to apply the settings.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are

Control (UI)	Default Value	Function
		available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

ArnoldGlobalSettings

The ArnoldGlobalSettings node is populated by the XML file located at **${KATANA_ROOT}/plugins/Resources/Arnold<version number>/GenericAssign**.



Note: The parameters that are available for this node are dependent on which version of Arnold you are using. As such, only renderer-agnostic parameters are listed. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with Arnold.

Connection Type	Connection Name	Function
Input	input	The 3D input to which you want to apply the settings.

ArnoldLiveRenderSettings

The ArnoldLiveRenderSettings node provides parameters for use by Arnold when live rendering. The parameters are defined in a GenericAssign **.xml** file, such as the **ArnoldGlobalSettings.xml** and **ArnoldObjectSettings.xml.**



Note: The parameters that are available for this node are dependent on which version of Arnold you are using. For more information on some of the other parameters, please refer to the documentation that ships with Arnold.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to specify the parameters used y Arnold when live rendering.

ArnoldOutputChannelDefine

Builds the parameters used by Arnold during render.



Note: The driverParameters change depending on the selected driver.



Note: The parameters that are available for this node are dependent on which version of Arnold you are using. As such, only renderer-agnostic parameters are listed. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with Arnold.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to specify the output channel type and settings.

Control (UI)	Default Value	Function
name	N/A	The name used by the RenderOutputDefine node for these output channel settings. This parameter usually matches the channel name. By allowing a different name to be sent to the RenderOutputDefine node, different filter and file types can be used for the same output variable.
type	RGBA	Sets the output channel type:

Control (UI)	Default Value	Function
		BYTE, INT, LONG, BOOL, FLOAT, DOUBLE, RGB, RGBA, ABSRGB, VECTOR, POINT, POINT2, STRING, POINTER, ARRAY, MATRIX, and ENUM

ArnoldShadingNode

The ArnoldShadingNode allows you to select an Arnold-specific shader to build complex shading networks. The last shading node in the shading network needs to be connected to a NetworkMaterial node in order to be connected to the 3D node graph and assigned to objects in the scene.



Note: The parameters that are available for this node are dependent on which version of Arnold you are using. As such, only renderer-agnostic parameters are listed. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with Arnold.

Control (UI)	Default Value	Function
name	ArnoldShadingNode	Determines the attribute identifier for this shader node beneath the 'material' attribute. This must be unique among all upstream nodes connected into a single NetworkMaterial node.
nodeType	N/A	Selects the available shader from the dropdown list. The parameters for each shader in the dropdown list are not included, as they are renderer-specific. Use the file browser or your studio's asset management browser to select the shader to use.
parameters	N/A	Once you've added a shader, the shader's parameters are populated under the **Parameter** group.
publicInterface		

Control (UI)	Default Value	Function
namePrefix	N/A	Specifies the name's prefix for the exposed parameter.
pagePrefix	N/A	Allows you to organize the shading node's exposed parameters in groups in the NetworkMaterial node's Material Interface.
nameRegExFind	N/A	Finds and deletes the name specified in **namePrefix** field.
nameRegExReplace	N/A	When used with **nameRegExFind**, finds and replaces the name with the name specified by **nameRegExReplace**.
pageRegExFind	N/A	Finds and deletes the name specified in **pagePrefix** field.
pageRegExReplace	N/A	When used with **pageRegExFind**, finds and replaces the name with the name specified by **pageRegExReplace**.
ArnoldShadingNode parameters continued		
Force Refresh	N/A	Reloads the shader file's information.

AttributeCopy

Copies an attribute from location(s) in the copyFrom scene to location(s) in the input scene. Attribute data is shared between copies, so it's cheap to copy large attributes like geometry.point.P.

This node traverses the copyFrom scene at location fromRoot, and the input scene at location toRoot. From these locations on, it expects to find identical hierarchy and location names. For each location, if the copyFrom location has the attribute specified by fromAttr, for example, geometry.point.P, the attribute is copied to the location specified by toAttr, for example, geometry.point.Pref on the input location.

The optional **toCEL** parameter allows you to filter the evaluation of this node. Only locations in the destination scene that match **toCEL** are evaluated. If **toCEL** is empty, all locations in the destination scene are evaluated.

Connection Type	Connection Name	Function
Input	input	The scene locations to which the specified attributes are copied.

Connection Type	Connection Name	Function
	copyFrom	The scene location from which the specified attributes are copied.

Control (UI)	Default Value	Function
fromRoot	/root/world	Defines the **copyFrom** location. The **fromRoot** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
fromAttr	N/A	Defines the attribute that is copied.
toRoot	/root/world	Defines the copyTo location. The **toRoot** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
toAttr	N/A	Defines the location where the attribute is copied to.
toCEL	N/A	Allows you to filter the evaluation of this node. Only locations in the destination scene that match **toCEL** are evaluated. If **toCEL** is empty, all locations in the destination scene are evaluated. The scene graph locations are specified using the Collection Expression Language (CEL). The **toCEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

AttributeEditor

The AttributeEditor node is used to edit specific attributes of objects in the scene graph.

Connection Type	Connection Name	Function
Input	in	The object whose attributes you want to edit.

Control (UI)	Default Value	Function
exclusivity	N/A	Exclusivity locks the interactive **Viewer** tab edits of a location to this node. The scene graph locations are specified using the Collection Expression Language (CEL). The **exclusivity** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
Existing Edits Table	N/A	Attributes dragged into the table are placed here for editing. You can resize this table by dragging on the right-corner.
Existing Edits Table		
Existing Edits	N/A	The name of the attribute, grouped under its scene graph location, is displayed in this column.
Index	N/A	When the edited attribute is a number or string array, the Index column controls which value is displayed in the **Value** column. Click in the column to popup a slider, which changes the index.
Value	N/A	The value of the edit is displayed in the **Value** column. Clicking on the value pops up a simple string or number field allowing you to change the value.

Control (UI)	Function
Existing Edits Table > [right-click menu]	
 Go To Location	Selects the scene graph location these edits affect.
 Disable Overrides	Disables the edit
 Enable Overrides	Enables the edit
Move Overrides To Selected Scenegraph Location...	Moves the override to the scene graph location currently selected.
Copy Overrides To Selected Scenegraph Location...	Copies the override to the scene graph location currently selected.
Delete Overrides	Deletes the override.

AttributeSet

This node is used for creating, modifying, or deleting scene graph attribute locations.

Connection Type	Connection Name	Function
Input	A	The place in the node graph where you want to amend attributes for a given scene graph location.

Control (UI)	Default Value	Function
mode	paths	Specifies the location to be overridden: • **paths**

Control (UI)	Default Value	Function
		• **CEL**
mode: paths		
paths	N/A	Sets the paths of the attribute. For example, **/root/world/geo**. The **paths** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
mode: CEL		
celSelection	N/A	Sets the attribute location to be overridden. The scene graph locations are specified using the Collection Expression Language (CEL). The **celSelection** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
AttributeSet parameters continued		
action	Create/Override	Describes the action to apply to the scene graph attribute: • **Create/Override** • **Delete** • **Force Default**
attributeName	N/A	The name of the attribute to set. To set the **attributeName**, either enter the name of the attribute into the text field, or drag and drop an attribute from the **Attributes** tab into the text field.

Control (UI)	Default Value	Function
		Note: When dropping a dragged attribute onto the **attributeName** field, the value is set to the full name of the dropped attribute, with names of ancestor groups separated by dots. For example, **xform.translate**.
action: Create/Override		
attributeType	double	The type of the attribute to set. To set the **attributeType**, select a data type from the drop down menu, or drag and drop an attribute from the **Attributes** tab into the drop down field. **Note:** When dropping a dragged attribute onto the **attributeType** field, the value is set to the name of the dropped attribute's type, for example, **float**. The drop-down menu contains the following data types: • integer • double • float • string • group
groupInherit	Yes	Decides whether or not implicitly-created groups are inherited lower in the scene graph hierarchy. For instance, creating **foo.bar** implicitly creates the group foo. This group is either inherited or not, depending on this parameter.
multisample	Yes	Can be used to enable or disable multi-sampling.
action: Create/Override: attributeType: integer, double, or float		
numberValue	0.0	Sets the override value.

Control (UI)	Default Value	Function
action: Create/Override: attributeType: string		
stringValue	N/A	Sets the override value.
action: Create/Override: attributeType: group		
applyAtLeaves	No	When assigning to a group with this parameter enabled, Katana adds the non-group attributes individually. This allows new assignments to mix with existing peers. When this parameter is disabled, the entire group-level is replaced and empty groups are not added.
groupValue	N/A	Collects a number of attributes into one group.
Drop Attributes Here	N/A	Middle-click and drag attributes from the **Attributes** tab to this hotspot to use that attribute.
action: Force Default		
groupInherit	Yes	Decides whether or not implicitly-created groups are inherited lower in the scene graph hierarchy. For instance, creating **foo.bar** implicitly creates the group foo. This group is either inherited or not, depending on this parameter.

BoundsAdjust

Allows you to adjust the bounding box of a geometry location.

Connection Type	Connection Name	Function
Input	input	The geometry location whose bounding box you want to modify.

Control (UI)	Default Value	Function
targetPath	N/A	Describes the location of the geometry who's bounding box is being adjusted. The **targetPath** parameter options are available in either a scene graph widget or dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
action	Strip Bounds	Sets the type of bounding adjustment to apply: • **Strip Bounds** - removes the bounding box from the geometry entirely. • **Pad Bounds (%)** - allows you to pad the bounding box by a user-defined percentage. • **Pad Bounds (local)** - allows you to pad the bounding box by an amount specified in units.
action: Strip Bounds		
stripAncestors	No	Specify whether or not to strip the bound attributes from the ancestor locations.
action: Pad Bounds (%) > padAmount		
percentage	0	Specify percentage to add to the original bounding box size. This is a keyable attribute.
adjustAncestors	No	Specify whether or not to expand the bound attributes of any ancestor location to account for the newly adjusted child location bounds.
when	immediate	Sets when the specified adjustment is applied: • **immediate** - pad the bounds immediately. • **deferred** - pad the bounds only at render time (more efficient). Padding is calculated in the PRMan plug-in, so the result is not visible in Katana even with implicit resolvers on.
action: Pad Bounds (Local) > padAmount		

Control (UI)	Default Value	Function
localSpace	0	Specifies the number of local space units to pad the bounding box.
adjustAncestors	No	Specify whether or not to expand the bound attributes of any ancestor location to account for the newly adjusted child location bounds.
when	immediate	Sets when the specified adjustment is applied: • **immediate** - pad the bounds immediately. • **deferred** - pad the bounds only at render time (more efficient). Padding is calculated in the PRMan plug-in, so the result is not visible in Katana even with implicit resolvers on.

CameraClippingPlaneEdit

Edits the camera near and far clipping attributes for a single camera.



Note: The default values change when initially connected to a camera.

Connection Type	Connection Name	Function
Input	input	The camera whose clipping attributes you want to modify.

Control (UI)	Default Value	Function
cameraLocation	/root/world/cam/camera	Describes the location of the camera. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.

Control (UI)	Default Value	Function
with location specified		
near	1 (see note)	Sets the near clipping plane for the specified camera.
far	10000 (see note)	Sets the far clipping plane for the specified camera.
claimExclusivity	No	When set to **No**, the camera is not controlled by another node in the scene graph and is effectively locked.

ConstraintCache

The ConstraintCache node caches the transform attributes of specified locations to disk for later use.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to cache transform attributes specified at specific locations.

Control (UI)	Default Value	Function
Fill Cache	n/a	Fills the cache with the transform matrix for a given location.
startFrame	1	Specifies from which frame to start caching.
endFrame	100	Specifies from which frame to stop caching.
locations	N/A	The scene graph location to where the cache is written. The **locations** parameter options are available by clicking **Add Locations or** ▾ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.

ConstraintListEdit

Adds locations to the **globals.constraintList** attribute at **/root/world**. This is useful for including constraints loaded from a deferred source, such as a look file. Only constraints on locations listed in the **globals.constraintList** are resolved at render time.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want constraints loaded from a deferred source to be resolved.

Control (UI)	Default Value	Function
locations	N/A	Sets the scene graph location(s) to add to the constraint list. The **locations** parameter options are available by clicking **Add Locations**. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
mode	add	Sets the constraint mode: • **add** - adds locations to the **globals.constraintList** at **/root/world**. • **remove** - removes locations from the **globals.constraintList** at **/root/world**.

FaceSetCreate

This node creates a set (or group) of faces in an existing mesh. This is useful in order to more easily re-select them later when making shader, attribute, and visibility assignments to a sub-set of faces on a single mesh.

Connection Type	Connection Name	Function
Input	in	The object or mesh for which you want to create additional faces.

Control (UI)	Default Value	Function
meshLocation	N/A	Describes the location of the mesh for which the set is created. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
faceSetName	faceset	Defines the name of the new set.
invertSelection	Disabled	When enabled, the faces that are not mentioned in the **selection** are used.
selection	N/A	Stores the list of faces as an array. Alternatively, stores the list of faces that are currently selected in the **Viewer** in this parameter by selecting ▼ > **Adopt Faces From Viewer**.

GenericOp

The GenericOp node allows you to run a named specific Op. This is particularly useful to run custom Ops written as plug-ins during development and for use within SuperTools and macros. For more information, refer to The Op API.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	The input ports you want to set for different parts of the node graph.

Control (UI)	Default Value	Function
opType	N/A	Specifies the type of the Op to run, for example, AttributeSet.
opArgs	N/A	Add a new opArgs parameters from the **Add** dropdown list. The available options are described in greater detail in Adding User Parameters. • **Number** • **String** • **Group** • **Number Array** • **String Array** • **Float Vector** • **Color, RGB** • **Color, RGBA** • **Button** • **Toolbar** • **TeleParameter** • **Node Drop Proxy**
multisampleOpArgs	Yes	Enables multi-sampling of the **opArgs** group parameter to Op Args.
addSystemOpArgs	No	If enabled, adds a 'system' opArg containing information from the graph state time slice, such as frame and shutter timings. This is only necessary for some Ops.
executionMode	immediate	Determines when the Op is run: • **immediate** - the script is run at the locations specified in the **applyWhere** parameter as it is evaluated at this node's point in the node graph. • **deferred** - the script is set up by this node but won't actually be run until a later node in the node graph, as specified by the **applyWhen** parameter.
executionMode: immediate		

Control (UI)	Default Value	Function
applyWhere	at locations matching CEL	Determines where the script is run: • **at all locations** - at all the locations in the node graph. • **at specific location** - 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.
resolveIds	N/A	When **executionMode** is set to **immediate**, specify a space-delimited list of strings to indicate that this script should only be resolved by Op resolvers that contain at least one matching "resolveId". This is an advanced feature for greater control over order of evaluation. A useful resolveIds is implicit_prepocess, which runs at the first implicit resolver, before other implicit resolvers, such as MaterialResolve and ConstraintResolve are run.
inputBehavior	by index	Controls how input ports on the node are mapped onto the inputs of the underlying Op. This parameter 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. When set to **only valid**, any valid input ports of the node are skipped when determining which inputs to pass to the underlying Op. When set to **by index**, all input ports of the node are represented in the list of inputs the Op sees; invalid inputs are represented as an Op of type **no-op**.
applyWhere: at specific locations		
location	/root/world/location	The location to create, if it doesn't already exist. Otherwise, the scene graph location at which the script is run.

Control (UI)	Default Value	Function
		For more information on the location widget parameters, see Common Parameter Widgets.
applyWhere: at locations matching CEL		
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
executionMode: deferred		
applyWhen	during op resolve	Determines when the script is run: • **during op resolve** - the script and its arguments are added as attributes to be executed later by an OpResolve node. If the Op isn't run by an explicit OpResolve node placed in the node graph, it is automatically run at render time by the **implicit resolvers**. • **during material resolve** - the script and its arguments are added as attributes under the **material.scenegraphLocationModifers** group attribute. This is primarily intended for material scene graph locations, allowing the material to specify a procedural process that is run at every location that the material is assigned to. The script is run as part of the material resolve process, and are executed just after the initial values for the material shader are created at the location. Examples of its use include randomizing or building procedural control over shader parameters. • **during katana look file resolve** - the script and its arguments are added as attributes under the Scene GraphLocationModifers group attribute and are evaluated by a LookFileResolve node or by the first

Control (UI)	Default Value	Function
		implicit resolver if no LookFileResolve node is present.
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
name	GenericOp	The name of the attribute group under which to store the Op type and args.
inputBehavior	by index	Controls how input ports on the node are mapped onto the inputs of the underlying Op. This parameter 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. When set to **only valid**, any valid input ports of the node are skipped when determining which inputs to pass to the underlying Op. When set to **by index**, all input ports of the node are represented in the list of inputs the Op sees; invalid inputs are represented as an Op of type **no-op**.
applyWhen: during op resolve		
recursiveEnable	No	When applying in a non-immediate state, enabling this results in the Op running at every location beneath the assigned locations. In general this is more efficient than using an equivalent recursive CEL statement. You can also override the **ops.*.recursiveEnable** attribute anywhere deeper in the tree to exclude evaluation at those locations. This is similar to the behavior of the visible or light linking attributes.

Control (UI)	Default Value	Function
recursiveEnable: yes		
disableAt	N/A	Execution is disabled for locations at or below this CEL statement. For large scene hierarchies, this is often less expensive than enabling evaluation at a larger number of leaf locations to avoid applying it to a smaller subset. The scene graph locations are specified for the **disableAt** parameter options by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
resolveIds	N/A	When **applyWhen** is set to **during op resolve**, you may specify a space-delimited list of strings to indicate that this Op should only be resolved by Op resolvers that contain at least one matching "resolveId." This is an advanced feature for greater control over order of evaluation. A useful resolveIds is implicit_prepocess, which runs at the first implicit resolver, before other implicit resolvers, such as MaterialResolve and ConstraintResolve are run.
applyWhen: during katana look file resolve		
recursiveEnable	No	When applying in a non-immediate state, enabling this results in the Op running at every location beneath the assigned locations. In general this is more efficient than using an equivalent recursive CEL statement. You can also override the **ops.*.recursiveEnable** attribute anywhere deeper in the tree to exclude evaluation at those locations. This is similar to the behavior of the visible or light linking attributes.
recursiveEnable: yes		

Control (UI)	Default Value	Function
disableAt	N/A	Execution is disabled for locations at or below this CEL statement. For large scene hierarchies, this is often less expensive than enabling evaluation at a larger number of leaf locations to avoid applying it to a smaller subset. The scene graph locations are specified for the **disableAt** parameter options by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

GroupMerge

The GroupMerge node is a SuperTool that creates a convenient interface for managing multiple nodes of the same type.

Within the GroupMerge interface, you can create any number of **nodes of the same type**, and these nodes are combined into a single output by merging them. The nodes are merged in the order they appear in the list.

This node is most often used to group nodes that have no input, but provide a scene graph location as an output. For example, the GroupMerge node could be used to manage multiple PrimitiveCreate nodes, and the output scene graph is all of the primitives merged together.



Note: When the GroupMerge node is first created, its type is not defined. You can create a node and then add it to the stacklist by **Shift**+middle-mouse and dragging from the **Node Graph** tab to the node's list in the **Parameters** tab. At that point, the GroupMerge is permanently typed as a group of the type of node that was dragged in.

Control (UI)	Function
	Creates a new node of the type associated with this node and adds it to the node list.
	Brings up a searchable list to aid in selection.
 / 	 Locks all nodes against editing.  Unlocks all nodes for editing.

Control (UI)	Function
[Right-click menu]	
Ignore Selected Entries	Disables the selected nodes.
 View At Location	Sets the current view node to the selected node
Delete Selected Entries	Deletes the selected node.
Duplicate Selected Entries	Duplicates the selected node, creating a new copy of both the node and matching its parameters.
Cut Selected Entries	Deletes the selected node and copies it to the clipboard.
Copy Selected Entries	Copies the selected node to the clipboard.
Paste	Paste the current clipboard node into this list.
Tearoff Parameters Of Selected Entries...	Create a new floating window with the parameters of this node on a tab inside.

HierarchyCopy

The HierarchyCopy node takes a scene graph location - or locations - and copies to a given destination location or locations.



Note: The HierarchyCopy node copies not just the source locations but all attributes pertaining to those source hierarchies, resulting in a potentially slow operation.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to copy the scene graph location(s) to a destination in the scene graph.

Control (UI)	Default Value	Function
pruneSource	No	When set to Yes, the source location, or locations, are pruned from the recipe.
copies		
sourceLocation	N/A	The scene graph location of the hierarchy to copy. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
destinationLocations	N/A	The scene graph location under which the copy - or copies - are created. The **destinationLocations** parameter options are available by clicking **Add Locations or** ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
makeNumberedCopies	No	When set to **Yes**, the scene graph locations created are sequentially numbered, and the option to make more than one copy is enabled.
makeNumberedCopies:Yes		
numCopies	1	When **makeNumberedCopies** is set to **Yes**, sets the number of copies to create.

Isolate

This node is used to remove objects from a scene. It allows you to select a set of locations to keep and it removes everything else. For example, you could isolate a character or two out of all the geometry in your scene.

The Isolate node cannot take a collection. You can however:

1. Right-click on the collection name in the **Scene Graph** tab and select **Find and Select...**.
2. From the **Parameters** tab of the Isolate node, select **isolateLocations** > **Add Locations** > **Replace with Scene Graph Selection**.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to remove specific object(s) from the scene.

Control (UI)	Default Value	Function
isolateLocations	N/A	This is a list of locations to keep while every other location is removed by the Isolate. For more information, refer to the Locations Widget Type in Common Parameter Widgets.
isolateFrom	/root/world/geo	This is the topmost location to remove from the scene. For example, if you set this to **/root/world/geo**, then nothing in **/root/world/lgt** or **/root/materials** is modified. This parameter allows you to scope the changes. To isolate a single shape from an entire character, set **isolateFrom** to the character path (for example, **/root/world/geo/**somecharacter), then set **isolateLocations** to the shape you'd like to keep. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.

Control (UI)	Default Value	Function
Enable secondary (inverse) output	disabled	When enabled, the secondary output provides a scene containing the scene graph locations that have been removed from the primary output. For example, in a scene containing the following locations: **/root/world/geo** **/root/world/geo/box** **/root/world/geo/circle** **/root/world/lgts** If **/root/world/geo/box** is isolated using isolateFrom **/root/world/geo**, the secondary output contains **/root/world/geo/circle**. **Note:** Any scene graph location that is a peer or ancestor of the isolationRoot is present on both outputs.

LightLink

The LightLink node manipulates the lightList attribute on the scene to perform selective lighting of objects. LightLink allows you to control which lights illuminate which objects, using a number of different modes.

Note: The GafferThree node uses a LightLink internally to provide light linking. The user interface there is substantially similar to the LightLink node.

Note: Light linking information is stored on the objects themselves in the lightList attribute. This stores the enable state of a light for each location in the scene. Visibility does not have any effect on lights, so a VisibilityAssign does not disable a light. LightLink is the best way to turn a light on or off by hand.

Connection Type	Connection Name	Function
Input	in	The light whose attributes you want to manipulate.

Control (UI)	Default Value	Function
effect	illumination	Determines whether the link is acting upon the light's illumination or shadow visibility of the specified objects: • **illumination** • **shadowvisibility** • **Custom** **Note:** Shadow visibility is only currently respected by Arnold renders.
action	off	Controls the LightLink node's behavior: • **on** - turn the selected lights on for the selected objects. Does nothing else. • **exclusive on** - turn the selected lights on for the selected objects. Also turn the selected lights off for all other objects. Use this to force the selected lights to only illuminate the selected objects, and nothing else. • **off** - turn the selected lights off for the selected objects. Does nothing else. • **exclusive off** - turn the selected lights off for the selected objects. Also turn the selected lights on for all other objects. Use this to force the selected lights to not illuminate the selected objects, but to illuminate everything else.

Control (UI)	Default Value	Function
		• **clear** - remove any local setting for light enable/disable for the selected objects; the inherited settings are used on these objects. • **delete** - the selected lights are removed from the light list for all objects in the scene. This is more than simply turning the lights off; they're removed from the list, and a LightListEdit is required to turn them on again. • **delete inverse** - the selected lights are the only lights left in the light list for all objects in the scene.
effect: custom		
customAttrName	custom	Specifies a custom attribute name to set on the lightList for your object scene graph locations. The value of the custom parameter becomes the attribute name, which is set on the object scene graph location for each light. When a LightLink node is used with a LightLinkEdit node, for instance, if the: • **light** parameter is set to **/root/world/lgt/spotlight**, • **custom** parameter is set to **myAttr**, • **off** CEL parameter is set to **/root/world/geo/pony**, Then • **/root/world/geo/pony** has an attribute named **lightList.root_world_lgt_spotlight.myAttr**, whose value is set to **0**.
objects	N/A	Sets the object(s) on which to operate. The scene graph locations are specified for the **objects** parameter options by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
lightMode	CEL	Controls how you specify which lights to operate on:

Control (UI)	Default Value	Function
		• **CEL** • **Paths** Paths are included for backward compatibility.
lightMode: CEL		
lights	N/A	When **lightMode** is set to **CEL**, this CEL statement is used to select the lights to operate on. The scene graph locations are specified using the Collection Expression Language (CEL). The **lights** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
lightMode: Paths		
lightPaths	N/A	When **lightMode** is set to **Paths**, this list of light path names is used as the set of lights to operate on. The **lightPaths** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.

LightLinkEdit

The LightLinkEdit node adjusts which objects are illuminated by a light. This node edits **lightList** attributes that were previously set by LightLink or LightLinkResolve nodes.

Note: Light linking information is stored on the object scene graph locations themselves in the lightList attribute. This stores the enable state of a light for each location in the scene.

Connection Type	Connection Name	Function
Input	in	The object whose lightList attributes you want to modify.

Control (UI)	Default Value	Function
clearUnmatched	disabled	The **clearUnmatched** parameter determines whether or not existing light linking attributes for this light are removed from locations that do not match the **on** or **off** expressions.
effect	illumination	Determines whether the link is acting upon the light's illumination or shadow visibility of the specified objects: • **illumination** • **shadow visibility** • **custom** **Note:** Shadow visibility is only currently respected by Arnold renders.
light	N/A	Specifies the scene graph location for the light you want to apply the lighting quality of the **effect** parameter from. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
effect: custom		
customAttrName	custom	Specifies a custom attribute name to set on the lightList for your object scene graph locations. The value of the custom parameter becomes the attribute name, which is set on the object scene graph location for each light. When a LightLink node is used with a LightLinkEdit node, for instance, if the: • **light** parameter is set to **/root/world/lgt/spotlight**, • **custom** parameter is set to **myAttr**,

Control (UI)	Default Value	Function
		• **off** CEL parameter is set to **/root/world/geo/pony**, Then • **/root/world/geo/pony** has an attribute named **lightList.root_world_lgt_spotlight.myAttr**, the value of which is set to **0**.
LightLinkEdit parameters continued		
on	N/A	Links matching locations to the effect specified in the **effect** parameter. The scene graph locations are specified using the Collection Expression Language (CEL). The **on** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
off	N/A	Disables the effect specified in the **effect** parameter for the matching locations. The scene graph locations are specified using the Collection Expression Language (CEL). The **off** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

LightLinkResolve

The LightLinkResolve node resolves the attributes, which the LightLinkSetup node sets on **/root/world.lightList**. The LightLinkSetup node defines an entry on the **lightList** containing CEL expressions defining On and Off locations for a particular light. The LightLinkResolve node resolves these CEL expressions into local attributes on any locations that match the criteria defined by these CEL expressions.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to resolve the attributes set up by LightLinkSetup.

LightLinkSetup

The LightLinkSetup node is an alternative to the LightLink node and sets the attributes on the lightList at the **/root/world** location instead of the object's scene graph location. It allows the node to set light linking options for locations that don't exist at that point in the node graph. These options are resolved by implicit resolvers or can be resolved manually by a LightLinkResolve node.

Connection Type	Connection Name	Function
Input	in	The light whose attributes you want to set on the lightList.

Control (UI)	Default Value	Function
clearUnmatched	disabled	When linking is resolved, the **clearUnmatched** parameter determines whether or not existing light linking attributes for this light are removed from locations that do not match the **on** or **off** expressions. The effect of this parameter is only visible in the **Attributes** tab when linking has been resolved, which means after a LightLinkResolve node or when Implicit Resolvers are active. Examines the **lightList** attribute on your linked objects to ensure that the attributes have been set correctly. If the attribute has been disabled, the value of the **enable** child attribute in the **lightList** attribute for your light is **0**; otherwise, the default enabled setting is **1**.
action	append linking	Determines how light linking settings from this node are

Control (UI)	Default Value	Function
	information	merged with settings in the incoming scene. If this **light** doesn't exist in the incoming scene, this option has no effect. • **append linking information** - the new attributes are appended to the incoming options. Where conflicts occur, the attributes that are set from this node are used. • **override linking information** - the linking options set in this node override information from the incoming scene options.
effect	illumination	Determines whether the link is acting upon the light's illumination or shadow visibility of the specified objects: • **illumination** • **shadow visibility** • **custom** **Note:** Shadow visibility is only currently respected by Arnold renders.
effect: custom		
customAttrName	custom	Specifies a custom attribute name to set on the lightList for your object scene graph locations. The value of the custom parameter becomes the attribute name that is set on the object scene graph location for each light. When a LightLinkSetup node is used with a LightLinkResolve node, for instance, if the: • **light** parameter is set to **/root/world/lgt/spotlight**, • **custom** parameter is set to **myAttr**, • **off** CEL parameter is set to **/root/world/geo/pony**, Then • **/root/world/geo/pony** has an attribute named **lightList.root_world_lgt_spotlight.myAttr**, whose value is set to **0**.

Control (UI)	Default Value	Function
		Note: If you've added a LightLinkSetup node only and have not linked it to a LightLinkResolve node, the attribute on **/root/world/geo/pony** is not set.
light	N/A	Specifies the scene graph location of the light you want to apply the lighting quality of the **effect** parameter from. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
initialState	use existing value	Determines the initial value for the specified effect in the light list entry for this light: • **use existing value** - the attribute uses the value of the attribute in the incoming scene if applicable. • **on** - sets the initial value to 1. • **off** - sets the initial value to 0.
LightLinkSetup parameters continued		
on	N/A	Links matching locations to the effect specified in the **effect** parameter. The scene graph locations are specified using the Collection Expression Language (CEL). The **on** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
off	N/A	Disables the effect specified in the **effect** parameter for the matching locations. The scene graph locations are specified using the Collection Expression Language (CEL). The **off** parameter options are available by clicking **Add Statements**.

Control (UI)	Default Value	Function
		For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

LightListEdit

This node adds locations to the lightList attribute at **/root/world**. This is useful for including lights whose loading is deferred. Only explicit paths are supported because this information is required at the start of rendering. LightListEdit can also be used to extracts lights from components and makes them renderable from a Look File.

Connection Type	Connection Name	Function
Input	in	The light whose lightList attributes you want to add locations to.

Control (UI)	Default Value	Function
locations	N/A	Sets the locations of lights from a path, either in the scene graph or the node graph. The **locations** parameter options are available by clicking **Add Locations** or ▼ dropdown menu. For more information, refer to the Scene Graph Location and Locations Widget Types in Common Parameter Widgets.
mode	add	Sets edit mode, though currently only **add** is available.
initialState	on	Determines whether the newly-added light locations are initially **on** or **off**.

LocationGenerate

The LocationGenerate node allows you to add a child location of a specific type to any locations matching a CEL statement.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to add a child location.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
name	loc	Specifies the child location name.
locationType	renderer procedural	Specifies the child location type.
addParentNameAsAttr	N/A	Optionally adds the basename of the matching CEL location at the specified attribute name. Typically, this parameter is used for locations to run renderer procedurals, if the name of the parent location needs to be passed to the renderer as a parameter.

LodGroupCreate

When pointed at a location, the children are assigned a level of detail (LOD) range for each node input. Each node input that requires an LOD range must be added as an additional input using the **inputs** > **Add** > **Add Input** menu option.

Connection Type	Connection Name	Function
Input	input0	The place in the node graph where you want to assign child locations to a level-of-detail range. Add numbered input ports (input1, input2) by adding inputs in the node's parameters.

Control (UI)	Default Value	Function
groupName	lod_group	The name of the **level-of-detail group** location that is created at the **hierarchyTargetLocation**.
hierarchyTargetLocation	/root/world/geo	The scene graph location where the **level-of-detail group** is placed. Each node input creates a **level-of-detail** location below this location, which stores the **lodRange** attributes for that input.
inputs	N/A	The parameter grouping for the node inputs.
inputs > Add		
Add Input	N/A	Menu option to add a new node input and create an additional **level-of-detail** location to store its scene.
inputs > input		
minVisible	0	When the bounding box is transformed to screen space, if its pixel count is less than the **minVisible**

Control (UI)	Default Value	Function
		parameter, the object is not displayed.
lowerTransition	0	When the bounding box is transformed to screen space, if its pixel count lies between the **minVisible** and **lowerTransition** parameters, the object is only part displayed.
upperTransition	9999999999999999774880982345603402956 8	When the bounding box is transformed to screen space, if its pixel count is between the **upperTransition** and **maxVisible** parameters, the object is only part displayed.
maxVisible	9999999999999999774880982345603402956 8	When the bounding box is transformed to screen space, if its pixel count is less than the **maxVisible** parameter, the object is not displayed.

LodSelect

This node removes all but one LOD (level-of-detail) location beneath the selected level-of-detail groups. The location to keep is selected based on one of three attributes, either:

- **by index** - select the level-of-detail location to keep based on its index in the child list of the level-of-detail group.
- **by tag** - select the **level-of-detail** location to keep based on its **info > componentLodTag** attribute.
- **by weight** - select the **level-of-detail** location to keep based on its **info > componentLodWeight** attribute. The **level-of-detail** location below the **level-of-detail group** location that is closest to the weight specified in the **selectionWeight** parameter is kept.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to remove all but one of the level-of-detail locations.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets. **Note:** CEL statement should match locations of type **level-of-detail group**.
mode	by index	Sets the method used to specify levels of detail: • **by index** • **by tag** • **by weight** **Note:** In cases of ambiguity, or where no groups match the criteria, the lowest index LOD group is selected (after all possible filtering has taken place) for the **by index** and **by tag** modes. The **by index** mode operates with strict matching, and produces an error if the chosen index does not exist.
mode is: by index		
selectionIndex	0	Sets the index of which LOD child to keep.
mode is: by tag		
selectionTag	hi	Sets the tag of which LOD child to keep.
mode is: by weight		
selectionWeight	1	Sets the weight to use while determining which children to keep.

LodValuesAssign

This node assigns level of detail (LOD) ranges to the child locations for all CEL statement matches. Each child location requires an LOD range and must be added using the **ranges > Add > Add Entry** menu option.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to assign level-of-detail ranges to child locations.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets. **Note:** CEL statement should match locations of type **level-of-detail group**.
ranges	N/A	The parameter grouping that holds the LOD ranges for each child location.
ranges > Add		
Add Entry	N/A	Menu option to add a new **ranges** parameter with **minVisible**, **lowerTransition**, **upperTransition**, and **maxVisible**. For every child location of the **level-of-detail group** location there should be a corresponding

		level-of-detail location and **ranges** parameter.
ranges > lod		
minVisible	0	When the bounding box is transformed to screen space, if its pixel count is less than the **minVisible** parameter, the object is not displayed.
lowerTransition	0	When the bounding box is transformed to screen space, if its pixel count lies between the **minVisible** and **lowerTransition** parameters, the object is only part displayed.
upperTransition	9999999999999999 7748809823456034029568	When the bounding box is transformed to screen space, if its pixel count is between the **upperTransition** and **maxVisible** parameters, the object is only part displayed.
maxVisible	9999999999999999 7748809823456034029568	When the bounding box is transformed to screen space, if its pixel count is less than the **maxVisible** parameter, the object is not displayed.

MaterialStack

MaterialStack node is a specialized GroupStack for organizing your scene Materials. To move a Material node that is outside the MaterialStack node to inside the stack, hold down the shift key and middle-mouse drag it in.

The Material in the stack are linked together, providing a single output by connecting them one after the other in serial, in the order in which they appear in the stack. Selecting Materials in the stack displays their controls on the right of the stack.

Merge

The Merge node allows you to combine multiple scenes into a single output scene. All objects in any of the input scenes are present in the output scene. If a location is present in more than one of the input scenes, then attribute values are taken from the left-most input, which has the location (however, the **Advanced**

Options allow more control over this). Merge is a very versatile node for collecting multiple elements into a scene for rendering.

Tips:

- A Merge node with a single input is effectively a no-Op node.
- Right-click on the Merge node in the **Node Graph** tab input ports to delete any unused ports.
- Right-click on a node while connecting a link in the **Node Graph** tab to display a pop-up menu of ports to connect to; this can be easier than hunting for a specific port on a Merge.
- Press the tilde key (~) while connecting a link in the **Node Graph** tab to connect to the left-most open port on the node, or add a new port if none are free.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	The input ports you want to set for different parts of the node graph.

Control (UI)	Default Value	Function
showAdvancedOptions	No	When set to **Yes**, the advanced parameters are available. These are normally only needed when doing something unusual or complex; merging two components together to form a single model is a common case, for example merging cloth and deforming geometry together. Typically, this use of the Merge node is hidden from the user inside a show macro so it's unlikely you'll need the advanced options.
showAdvancedOptions: Yes		
advanced		
sumBounds	No	When enabled, bound attributes are queried for each relevant input location and the total results are used. The output bounding box at each location is expanded to be large enough to contain all the

Control (UI)	Default Value	Function
		inputs at that location. This is important when merging renderable geometry together inside of components.
preserveWorldSpaceXform	No	When enabled, all inherited xform attributes (preceded by an origin statement) are applied at each location whose source input differs from that of its parent. This is only necessary in exceptional situations where there are conflicting transformations on overlapping locations of the merge inputs. Basically, this forces some locations to ignore their parent transforms so that they appear in the correct location in the scene. This is most commonly used when merging deforming geometry into a component, because the deforming geometry may have different transforms on locations shared with the non deforming geometry. If the result of the merging has objects that seem to be in the wrong position, try this option as a possible solution.
preserveInheritedAttributes	N/A	Displays a list of attribute names for which inheritance should be preserved when choosing between inputs of the Merge. Whenever a child location's source input differs from that of its parent, these attributes are queried globally and applied locally to the child location.
preferredInputAttributes	N/A	Displays a list of attribute names and indices of inputs for which the preferred value of an attribute should be read. These are exceptions to the general rule of leftmost input wins. For the listed attribute, a given input is given 'first crack' at providing the attribute in the result before the general rule is used. This is often used when merging two versions of a component to form a single output model; the first input provides most of the attributes, but a second input might provide correctly deformed geometry or other attributes that should be used in preference to

Control (UI)	Default Value	Function
		the first input. Again, this is typically rolled into a show macro, so it's unlikely you'll need to work with this setting directly.
mergeGroupAttributes	N/A	Used to specify and merge group attributes of the same name from different input scenes. For example, this parameter can be used to merge different types of materials, or materials for different renderers.
inputs		
inputs	N/A	Allows you to name the inputs on the Merge node.

NetworkMaterial

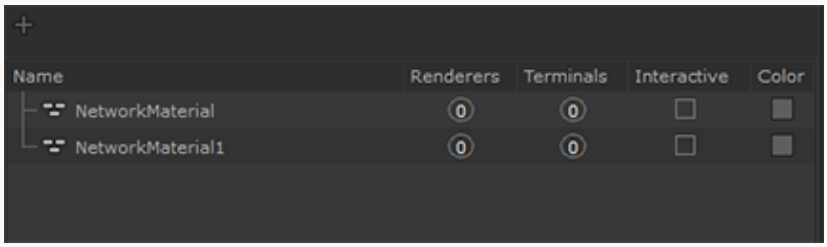
The NetworkMaterial node creates a material location for shading nodes that are connected as inputs, in order for the material to be assigned to objects in the scene. You can assign the material locations that are created by a NetworkMaterial node to a location with MaterialAssign node.

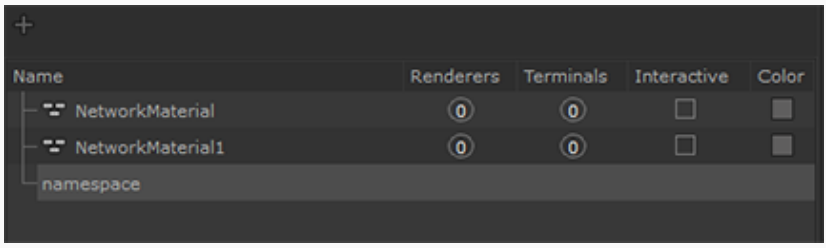
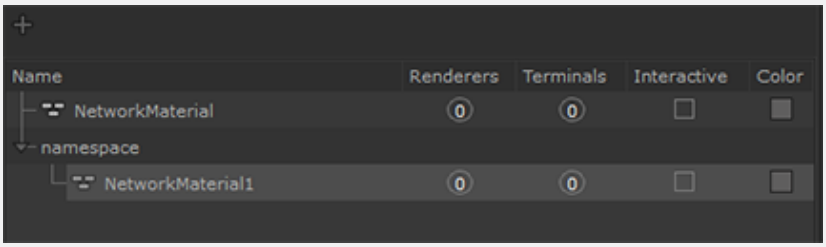
Control (UI)	Default Value	Function
name	NetworkMaterial	Specifies the name of the location.
namespace	N/A	Creates a sub-location for the network material in the **Scene Graph** tab.
Add Terminal ▼	N/A	Specifies which input to expect from the shading nodes. For instance, a dlSurface or dlDisplacement port will accept input from dlShadingNodes.
Material Interface ⚙ **or right-click on interface table**		
Rebuild with Current State	N/A	Updates the information of the Material Interface.
Remove Selected Local Definition...	N/A	Removes the selected shading node's exposed parameters.

Control (UI)	Default Value	Function
Remove All Local Definitions...	N/A	Removes all selected shading node's exposed parameters.

NetworkMaterialCreate

The NetworkMaterialCreate node has been designed to contain your material network. The node features a left-to-right workflow and a new shading node design, which enables you to work more efficiently, making building and editing materials as quick and simple as possible. It holds the function of one or more NetworkMaterial nodes as well as the NetworkMaterialInterfaceControls node.

Control (UI)	Default Value	Function
rootLocation	/root/materials	Defines the scene graph location where the material locations are created. The parameter options are available in either the scene graph widget or drop-down menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type.
➕ Add NetworkMaterial/ Add Namespace	N/A	Click to add a new NetworkMaterial location or Namespace. • **Add NetworkMaterial** - Add a new NetworkMaterial location, accessible from within this NetworkMaterialCreate node.  Added a new NetworkMaterial called **NetworkMaterial1** • **Add Namespace** - Add a new namespace to the Scene Graph under which NetworkMaterial locations can be grouped.

Control (UI)	Default Value	Function
		 Added a new Namespace called **namespace** **Note:** To learn more about multi-NetworkMaterial workflows in a NetworkMaterialCreate node, see Multiple NetworkMaterials with NetworkMaterialCreate.
Material Scenegraph	N/A	Add, remove and organize NetworkMaterials and Namespaces. • **Name** - To change the name of a NetworkMaterial or Namespace, you can double-click, or select the NetworkMaterial or Namespace and press **Enter** on the keyboard. You can then type a new name. • **Renderers** - View the number of renderers connected to each NetworkMaterial. • **Terminals** - View the number of terminals connected to each NetworkMaterial. • **Interactive** - When checked, you can drag objects in the Viewer and Katana retains the information from the Viewer. • **Color** - Set the color of a NetworkMaterial which is then applied to the terminal sidebar within the NetworkMaterialCreate node. Use the middle-mouse button to click and drag NetworkMaterials and Namespaces to organize them. 

Control (UI)	Default Value	Function
		Middle-mouse drag **NetworkMaterial1** to place it under the **namespace** **Note:** You can place NetworkMaterials and Namespaces underneath other Namespaces but not underneath other NetworkMaterials.

Node Parameters

Control (UI)	Default Value	Function
parameters	N/A	Promoted parameters within the NetworkMaterialCreate node appear here. This section remains empty if no parameters have been promoted. **Note:** For more information on how to promote parameters, see Node Parameters and Interface Controls documentation.

Interface Controls

Control (UI)	Default Value	Function
Add Node button	N/A	Click to add a new Interface Control and open its parameters.
Filter	N/A	Filter the list of Interface Controls by name or type. Type in the text field to start filtering. A list of matching controls are displayed. Click the **Select All Matching** button to display the parameters for

Control (UI)	Default Value	Function
		all matching Interface Controls.
Disable Parameter Display	Disabled	Toggles the display of the parameters.
Fit to Width	Disabled	Adjusts the width of the Interface Control list so the full length of the control names are visible.
Fit to Height	Disabled	Adjusts the height of the Interface Control list so all controls are visible.
Note: The following parameters are only visible once an Interface Control has been created and is selected.		
state	visibility	The state of the parameter or page for the control to affect. • **visibility** - Depending on the condition, displays or hides the parameter or page specified in the **targetName** parameter. • **lock** - Depending on the condition, locks or unlocks the parameter or page specified in the **targetName** parameter, making any edits impossible if locked.
targetType	parameter	Select whether to apply the condition on either: • **parameter** - The operation affects a single parameter. • **page** - The operation affects a page containing one or more parameters.
targetName	N/A	Specifies the name of the chosen parameter or page. **Note:** The name must be identical to the one displayed in the NetworkMaterial node's Material Interface.
definitionStyle	operator tree	Selects how to set up the condition:

Control (UI)	Default Value	Function
		• **operator tree** - Allows you to set up conditions using an operator tree. • **conditional state expression** - Allows you to set up conditions using one or several expressions.
When **definitionStyle: operator tree**		
operators		
op	and	Select which expression operator to use in the operator tree: • **and** - The resulting expression is satisfied only if all of the child expressions are satisfied. • **or** - The resulting expression is satisfied if at least one of the child expressions is satisfied.
ops		
Add	N/A	Select an op: • **contains** - Evaluates if the condition is true by testing if the parameter or page values contain the values set in the expression. • **doesNotContain** - Evaluates if the condition is true by testing if the parameter or page values do not contain the values set in the expression. • **endsWith** - Evaluates if the condition is true by testing if the parameter or page values end with the values set in the expression. • **equalTo** - Evaluates if the condition is true by testing if the parameter or page values are equal to the values set in the expression. • **greaterThan** - Evaluates if the condition is true by testing if the parameter or page values are greater than the values set in the expression. • **greaterThanOrEqualTo** - Evaluates if the condition is true by testing if the parameter or page values are greater than or equal to the values set in the expression. • **in** - Evaluates if the condition is true by testing if the

Control (UI)	Default Value	Function
		parameter or page values are in the values (separated by a pipe with no spaces) set in the expression. • **lessThan** - Evaluates if the condition is true by testing if the parameter or page values are less than the values set in the expression. • **lessThanOrEqualTo** - Evaluates if the condition is true by testing if the parameter or page values are less than or equal to the values set in the expression. • **notEqualTo** - Evaluates if the condition is true by testing if the parameter or page values are not equal to the values set in the expression. • **notIn** - Evaluates if the condition is true by testing if the parameter or page values are not in the values (separated by a pipe with no spaces) set in the expression. • **numChildrenEqualTo** - Evaluates if the condition is true by testing if the number of children in the target group parameter is equal to the number of children specified in the parameter or page. • **numChildrenGreaterThanOrEqualTo** - Evaluates if the condition is true by testing if the number of children in the target group parameter is greater than or equal to the number of children specified in the parameter or page. • **regex** - Evaluates if the condition is true by testing if the parameter or page values match the values set in the regular expression. • **and** - Specifies if you want to compare the parameter or page values to another set of values. It uses all the expressions to evaluate the condition. • **or** - Specifies if you want to compare the parameter or page values to another set of values. It uses only one of the expressions to evaluate the condition.

Note: The following parameters are only visible once an operator has been selected from the **Add** menu.

op	N/A	The chosen op from the **Add** menu.

Control (UI)	Default Value	Function
path	N/A	Specifies the path of the parameter or page to evaluate.
value	N/A	Specifies the values to compare the parameter or page values with, in order to evaluate if the condition is true.

Material Interface

Control (UI)	Default Value	Function
Name	N/A	A list displaying all promoted parameters, organized in the same way as they were grouped when promoted.
Source	N/A	The path to each different parameter.



Note: For more information on the uses of the Material Interface, see Node Parameters and Interface Controls documentation.

NetworkMaterialEdit

The NetworkMaterialEdit nodes allow artists to edit network materials that have been created using NetworkMaterialCreate nodes or brought in through lookfiles, by adding or removing shading nodes from an existing network, or by modifying any of the parameters of shading nodes in an existing Network. NetworkMaterialEdit nodes hold the functionality of the existing NetworkMaterialParameterEdit and NetworkMaterialSplice node types, but in a UI that is visually representative of how the material was originally authored.

Node Parameters

Control (UI)	Default Value	Function
sceneGraphLocation	N/A	Type or drag the NetworkMaterial location from the Scene Graph to this field to specify which Network Material to make edits to. **Note:** The NetworkMaterialEdit node is designed to complement the NetworkMaterialCreate node. Only NetworkMaterial locations in your Scene Graph which have been created using a NetworkMaterialCreate, or lookfiles baked from a NetworkMaterialCreate node, can be edited using the NetworkMaterialEdit node.
makeInteractive	No	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
parameters	N/A	This is where promoted parameters from with the NetworkMaterialCreate node appear. This section remains empty if no parameters have been promoted. **Note:** For more information on how to promote parameters, see Node Parameters and Interface Controls documentation.

Interface Controls

Control (UI)	Default Value	Function
Add Node button	N/A	Click to add a new Interface Control and open its parameters.

Control (UI)	Default Value	Function
Filter	N/A	Filter the list of Interface Controls by name or type. Type in the text field to start filtering. A list of matching controls are displayed. Click the **Select All Matching** button to display the parameters for all matching Interface Controls.
Disable Parameter Display	Disabled	Toggles the display of the parameters.
Fit to Width	Disabled	Adjusts the width of the Interface Control list so the full length of the control names are visible.
Fit to Height	Disabled	Adjusts the height of the Interface Control list so all controls are visible.
Note: The following parameters are only visible once an Interface Control has been created and is selected.		
state	visibility	The state of the parameter or page for the control to affect. • **visibility** - Depending on the condition, displays or hides the parameter or page specified in the **targetName** parameter. • **lock** - Depending on the condition, locks or unlocks the parameter or page specified in the **targetName** parameter, making any edits impossible if locked.
targetType	parameter	Select whether to apply the condition on either: • **parameter** - The operation affects a single parameter. • **page** - The operation affects a page containing one or more parameters.
targetName	N/A	Specifies the name of the chosen parameter or page.

Control (UI)	Default Value	Function
		Note: The name must be identical to the one that displays in the NetworkMaterial node's Material Interface.
definitionStyle	operator tree	Selects how to set up the condition: • **operator tree** - Allows you to set up conditions using an operator tree. • **conditional state expression** - Allows you to set up conditions using one or several expressions.
When **definitionStyle: operator tree**		
operators		
op	and	Select which expression operator to use in the operator tree: • **and** - The resulting expression is satisfied only if all of the child expressions are satisfied. • **or** - The resulting expression is satisfied if at least one of the child expressions is satisfied.
ops		
Add	N/A	Select an op: • **contains** - Evaluates if the condition is true by testing if the parameter or page values contain the values set in the expression. • **doesNotContain** - Evaluates if the condition is true by testing if the parameter or page values do not contain the values set in the expression. • **endsWith** - Evaluates if the condition is true by testing if the parameter or page values end with the values set in the expression. • **equalTo** - Evaluates if the condition is true by testing if the parameter or page values are equal to the values set in the expression. • **greaterThan** - Evaluates if the condition is true by testing if

Control (UI)	Default Value	Function
		the parameter or page values are greater than the values set in the expression. • **greaterThanOrEqualTo** - Evaluates if the condition is true by testing if the parameter or page values are greater than or equal to the values set in the expression. • **in** - Evaluates if the condition is true by testing if the parameter or page values are in the values (separated by a pipe with no spaces) set in the expression. • **lessThan** - Evaluates if the condition is true by testing if the parameter or page values are less than the values set in the expression. • **lessThanOrEqualTo** - Evaluates if the condition is true by testing if the parameter or page values are less than or equal to the values set in the expression. • **notEqualTo** - Evaluates if the condition is true by testing if the parameter or page values are not equal to the values set in the expression. • **notIn** - Evaluates if the condition is true by testing if the parameter or page values are not in the values (separated by a pipe with no spaces) set in the expression. • **numChildrenEqualTo** - Evaluates if the condition is true by testing if the number of children in the target group parameter is equal to the number of children specified in the parameter or page. • **numChildrenGreaterThanOrEqualTo** - Evaluates if the condition is true by testing if the number of children in the target group parameter is greater than or equal to the number of children specified in the parameter or page. • **regex** - Evaluates if the condition is true by testing if the parameter or page values match the values set in the regular expression. • **and** - Specifies if you want to compare the parameter or page values to another set of values. It uses all the expressions to evaluate the condition. • **or** - Specifies if you want to compare the parameter or page values to another set of values. It uses only one of the

Control (UI)	Default Value	Function
		expressions to evaluate the condition.



Note: The following parameters are only visible once an operator has been selected from the **Add** menu.

op	N/A	The chosen op from the **Add** menu.
path	N/A	Specifies the path of the parameter or page to evaluate.
value	N/A	Specifies the values to compare the parameter or page values with, in order to evaluate if the condition is true.

Material Interface

Control (UI)	Default Value	Function
Name	N/A	A list displaying all promoted parameters, organized in the same way as they were grouped when promoted.
Source	N/A	The path to each different parameter.



Note: For more information on the uses of the Material Interface, see Node Parameters and Interface Controls documentation.

NetworkMaterialInterfaceControls

The NetworkMaterialInterfaceControls node allows you to apply a visibility or a lock condition on one or several parameters exposed by a NetworkMaterial node. This node works on a network material location, which can come from a NetworkMaterial node or a Look File.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to apply a visibility or lock condition on the parameters exposed by a NetworkMaterial node.

Control (UI)	Default Value	Function
materialLocation	N/A	Specifies the scene graph location path of the network material to be modified. The **materialLocation** parameter options are available by clicking the dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
state	visibility	Selects a condition state to apply: • **visibility** - depending on the condition, displays or hides the parameter or page specified in the **targetName** parameter. • **lock** - depending on the condition, locks or unlocks the parameter or page specified in the **targetName** parameter, making any edits impossible if locked.
targetType	parameter	Selects whether to apply the condition on either: • **parameter** - applies on a single parameter. • **page** - applies on a set of parameters grouped into a page.
targetName	N/A	Specifies the name of the chosen parameter or page. **Note:** The name must be identical to the one that displays in the NetworkMaterial node's Material Interface.
definitionStyle	operator tree	Selects how to set up the condition: • **operator tree** - allows you to set up conditions

Control (UI)	Default Value	Function
		using an operator tree. • **conditional state expression** - allows you to set up conditions using one or several expressions.
When definitionStyle: operator tree		
operators		
op	and	Selects which expression operator to use in the operator tree: • **and** - the resulting expression is satisfied only if all of the child expressions are satisfied. • **or** - the resulting expression is satisfied if at least one of the child expressions is satisfied.
ops > Add **or** **once an op has been added, [Op]**		
contains	N/A	Evaluates if the condition is true by testing if the parameter or page values contain the values set in the expression.
doesNotContain	N/A	Evaluates if the condition is true by testing if the parameter or page values do not contain the values set in the expression.
endsWith	N/A	Evaluates if the condition is true by testing if the parameter or page values end with the values set in the expression.
equalTo	N/A	Evaluates if the condition is true by testing if the parameter or page values are equal to the values set in the expression.
greaterThan	N/A	Evaluates if the condition is true by testing if the parameter or page values are greater than the values set in the expression.

Control (UI)	Default Value	Function
greaterThanOrEqualTo	N/A	Evaluates if the condition is true by testing if the parameter or page values are greater than or equal to the values set in the expression.
in	N/A	Evaluates if the condition is true by testing if the parameter or page values are in the values (separated by a pipe with no spaces) set in the expression.
lessThan	N/A	Evaluates if the condition is true by testing if the parameter or page values are less than the values set in the expression.
lessThanOrEqualTo	N/A	Evaluates if the condition is true by testing if the parameter or page values are less than or equal to the values set in the expression.
notEqualTo	N/A	Evaluates if the condition is true by testing if the parameter or page values are not equal to the values set in the expression.
notIn	N/A	Evaluates if the condition is true by testing if the parameter or page values are not in the values (separated by a pipe with no spaces) set in the expression.
numChildrenEqualTo	N/A	Evaluates if the condition is true by testing if the number of children in the target group parameter is equal to the number of children specified in the parameter or page.
numChildrenGreaterThan OrEqualTo	N/A	Evaluates if the condition is true by testing if the number of children in the target group parameter is greater than or equals to the number of children specified in the parameter or page.
regex	N/A	Evaluates if the condition is true by testing if the parameter or page values match the values set in the regular expression.
and	N/A	Specifies if you want to compare the parameter or

Control (UI)	Default Value	Function
		page values to another set of values. It uses all the expressions to evaluate the condition.
or	N/A	Specifies if you want to compare the parameter or page values to another set of values. It uses only one of the expressions to evaluate the condition.
Once an Op has been added:		
path	N/A	Specifies the path of the parameter or page to evaluate.
value	N/A	Specifies the values to compare the parameter or page values with, in order to evaluate if the condition is true.
When definitionStyle: conditional state expression		
expression	N/A	Specifies the expression to use to apply a visibility or a lock condition on the exposed parameter(s).

NetworkMaterialMultiSplice

The **NetworkMaterialMultiSplice** node is an extension from the NetworkMaterialSplice node. The **NetworkMaterialMultiSplice** provides additional fields for reference material, allows CEL expressions to target multiple materials, and gives you the ability to connect wildcarded node names.

This spliced material can then further be edited in a downstream NetworkMaterialEdit node. Viewing the material downstream will include the spliced nodes.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to read in an existing network material node and its corresponding shading network.

Control (UI)	Default Value	Function
materialsToEdit		
referenceMaterial	n/a	Indicate the material to appear in the node pop-up window.
spliceMaterial	n/a	Include further materials to receive the spliced information.
errorOnFail	Yes	If enabled, any errors applying the material splice will generate an error on the target material location. If set to 'No', any errors will be handled silently and the material will remain unchanged.
warnOnMultipleMatches	Yes	If enabled, a warning will be raised on the material location in the event that a wildcard node name match has applied to multiple materials. Otherwise, multiple connections will be made, but without warning.
connections	n/a	Specifies which nodes you want to connect within the shading node.
disconnections	n/a	Specifies which nodes you want to disconnect within the shading node.

NetworkMaterialParameterEdit

The NetworkMaterialParameterEdit node allows you to edit a shading node's parameters in a non-destructive way.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to edit a shading node's parameters.

Control (UI)	Default Value	Function
name	NetworkMaterialParameterEdit	Specifies the name of the location.
action	edit existing location	Determines the node's behavior: • **edit existing location** - specifies the NetworkMaterial scene graph location. • **inherit from existing material location** - creates a sub-location that inherits the material from the parent NetworkMaterial location.
When action is: edit existing location		
Material to Edit > location	N/A	Specifies the location. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
When action is: inherit from existing material location		
inheritsFrom > location	N/A	Specifies the location. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
nodes		
nodes > Add ▼	N/A	Allows you to add the shading nodes connected to the NetworkMaterial node.

NetworkMaterialSplice

The NetworkMaterialSplice node allows you to make changes to an existing NetworkMaterial node in a non-destructive way by connecting or inserting new shading nodes within the shading network. You can also

disconnect existing shading nodes within the shading network and rewire the NetworkMaterial node in different ways.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to read in an existing network material node and its corresponding shading network.
	append	The shading network you want to append to the existing shading network in the **in** port.

Control (UI)	Default Value	Function
name	NetworkMaterialSplice	Specifies the name of the node.
action	edit existing location	Determines the node's behavior: • **edit existing location** - makes modifications to an existing network material. • **inherit from existing material location** - creates a sub-location that inherits the material from the parent NetworkMaterialSplice location.
When action is: edit existing location		
edit > location	N/A	Specifies the location. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
When action is: inherit from existing material location		
inheritsFrom > location	N/A	Specifies the location. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.

Control (UI)	Default Value	Function
inputs > append		
action	connect	Allows you to select how the new shading nodes connected to the **append** input port are to be spliced into the network material: • **connect** - allows you to connect the new nodes to one of the input connections of a shading node in the original network material. • **insert** - allows you to insert the new nodes between two shading nodes within the shading network.
When action is: connect		
connectToNode	N/A	Specifies which shading node in the original network to connect the nodes to.
connectToPort	N/A	Specifies which port on the shading node we want to connect the new nodes to.
when action is: insert		
connectToNode	N/A	Specifies the point in between two consecutive shading nodes in the original material where you want to insert the new shading nodes.
connectToPort	N/A	Specifies which port to connect to.
reconnectToNode	N/A	Specifies the new shading node that you want the original connection to be re-connected to.
reconnectToPort	N/A	Specifies which port to reconnect to.
extraConnections > Add		
c > connectFromNode	N/A	Specifies the shading node in the original network that you want to create a new connection from.
c > connectFromPort	out	Specifies which output port on the **connectFromNode** to connect from.
c > connectToNode	N/A	Specifies the node in the original shading network that

Control (UI)	Default Value	Function
		you want to add a new connection to.
c > connectToPort	N/A	Specifies which port to connect to.
disconnections >Add		
d > node	N/A	Specifies which node you want to disconnect within the shading node.
d > port	N/A	Specifies which port you want to disconnect within the shading node.

OpResolve

This node resolves deferred Ops, such as OpScripts.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where deferred Ops are resolved.

Control (UI)	Default Value	Function
resolveWithIds	all	When an Op has been deferred to run **during op resolve** you must specify the resolveID. If the resolveWithIds option is set to **all**, it processes all the ops set to run **during op resolve**. If the option is set to **selected**, it only processes those that have at least one resolveId that matches the values set in the **specifiedResolveIds** field.
resolveWithIds: specified		
specifiedResolveIds	N/A	A space-separated list of resolveIds to resolve.

OpScript

A Lua-based interface to the Op API. For more information on the OpScript interface, see **Help** > **Documentation**.

Connection Type	Connection Name	Function
Input	i0	The place in the node graph where you want to specify Ops with the Op API.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
script	N/A	• **Edit in <editor>...** - opens an external editor, as set up in the **Preferences** under **externalTools** > **Editor**, for editing the OpScript node's Lua script without blocking Katana's user interface. Katana monitors the text files you are editing, and when it detects that they have changed, it updates the OpScript node accordingly. • **lua** - contains the lua script to run.
executionMode	immediate	Determines when the script is run: • **immediate** - the script is run at the locations specified in the **applyWhere** parameter as it is

Control (UI)	Default Value	Function
		evaluated at this node's point in the node graph. • **deferred** - the script is set up by this node but won't actually be run until a later node in the node graph, as specified by the **applyWhen** parameter.
multisampleUserOpArgs	Yes	Enables multi-sampling of the **opArgs** user parameter to Op Args.
Display as multi-input	Disabled	If enabled, allows multiple inputs to be connected to this OpScript node.
When executionMode is: immediate		
applyWhere	at locations matching CEL	Determines where the script is run: • **at all locations** - at all the locations in the node graph. • **at specific location** - 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.
inputBehavior	by index	Controls how input ports on the node are mapped onto the inputs of the underlying Op. This parameter 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. When set to **only valid**, any invalid input ports of the node are skipped when determining which inputs to pass to the underlying Op. When set to **by index**, all input ports of the node are represented in the list of inputs the Op sees; invalid inputs are represented as an Op of type no-op.
When applyWhere is: at specific locations		

Control (UI)	Default Value	Function
location	/root/world/location	The location to create, if it doesn't already exist. Otherwise, the scene graph location at which the script is run. The **location** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
When executionMode is: deferred		
applyWhen	during op resolve	Determines when the script is run: • **during op resolve** - the script and its arguments are added as attributes to be executed later by an OpResolve node. If the Op isn't run by an explicit OpResolve node placed in the node graph, it is automatically run at render time by the **implicit resolvers**. • **during material resolve** - the script and its arguments are added as attributes under the **material.ops** group attribute. This is primarily intended for material scene graph locations, allowing the material to specify a procedural process that is run at every location that the material is assigned to. The script is run as part of the material resolve process, and is executed just after the initial values for the material shader are created at the location. Examples of its use include randomizing or procedural control over shader parameters. • **during katana look file resolve** - the script and its arguments are added as attributes under the **ops** group attribute and are evaluated by a LookFileResolve node or by the first implicit resolver if no LookFileResolve node is present.

Control (UI)	Default Value	Function
modifierNameMode	node name	Deferred OpScripts are added as group attributes within the **ops** group. By default, 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. In nearly all cases, that doesn't matter. For cases in which it does, you can specify a fixed name to use.
When modifierNameMode is: specified		
modifierName	modifier	Sets the name of the attribute group beneath **ops** to use for describing this deferred script.
When executionMode is: deferred, during op resolve		
resolveIds	N/A	Specify a space-delimited list of strings to indicate that this script should only be resolved by Op resolvers that contain at least one matching "resolveId." This is an advanced feature for greater control over order of evaluation. A useful resolveIds is implicit_prepocess, which runs at the first implicit resolver, before other implicit resolvers, such as MaterialResolve and ConstraintResolve are run.
recursiveEnable	No	When applying in a non-immediate state, enabling this results in the script running at every location beneath the assigned locations. In general this is more efficient than using an equivalent recursive CEL statement. You can also override the **ops.*.recursiveEnable** attribute anywhere deeper in the tree to exclude evaluation at those locations. This is similar to the behavior of the visible or light linking attributes.
When recursiveEnable is: yes		
disableAt	N/A	Execution is disabled for locations at or below this

Control (UI)	Default Value	Function
		CEL statement. For large scene hierarchies, this is often less expensive than enabling evaluation at a larger number of leaf locations to avoid applying it to a smaller subset. The scene graph locations are specified for the **disableAt** parameter options by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
When executionMode is: deferred, during Katana look file resolve		
recursiveEnable	No	When applying in a non-immediate state, enabling this results in the script running at every location beneath the assigned locations. In general this is more efficient than using an equivalent recursive CEL statement. You can also override the ops.*.recursiveEnable attribute anywhere deeper in the tree to exclude evaluation at those locations. This is similar to the behavior of the visible or light linking attributes.
When recursiveEnable is: Yes		
disableAt	N/A	Execution is disabled for locations at or below this CEL statement. For large scene hierarchies, this is often less expensive than enabling evaluation at a larger number of leaf locations to avoid applying it to a smaller subset. The scene graph locations are specified for the **disableAt** parameter options by clicking **Add Statements**.

Control (UI)	Default Value	Function
		For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

PrmanGlobalSettings

This is for changing anything that broadly comes under the heading of RenderMan options.

Note: The parameters available for this node are dependent on which version of RenderMan for Katana you are using. As such, only the renderer-agnostic parameters are listed below. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with RenderMan.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to edit the parameters used by PRMan.

PrmanObjectSettings

The purpose of this node is to set PRMan attributes at levels of the scene graph hierarchy described by the given CEL statement.

Note: The parameters available for this node are dependent on which version of RenderMan for Katana you are using. As such, only the renderer-agnostic parameters are listed below. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with RenderMan.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to set PRMan attributes.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

PrmanOutputChannelDefine

The PrmanOutputChannelDefine allows you to define custom output channels, so that the final render can be split into separate elements. These customs channels (AOVs) can then be manipulated in a compositing tool.



Note: The parameters available for this node are dependent on which version of RenderMan for Katana you are using. As such, only the renderer-agnostic parameters are listed below. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with RenderMan.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to define custom output channels to be used by PRMan at the final render.

Control (UI)	Default Value	Function
name	N/A	The name used by the RenderOutputDefine node for these output channel settings. This parameter usually matches the channel name.
type	varying float	Sets the output channel type: BYTE, INT, LONG, BOOL, FLOAT, DOUBLE, RGB, RGBA, ABSRGB, VECTOR, POINT, POINT2, STRING, POINTER, ARRAY, MATRIX, and ENUM

PrmanShadingNode

The PrmanShadingNode allows you to select a RenderMan-specific shader to build complex shading networks. The last shading node in the shading network needs to be connected to a NetworkMaterial node in order to be connected to the 3D node graph and assigned to objects in the scene.



Note: The parameters available for this node are dependent on which version of RenderMan for Katana you are using. As such, only the renderer-agnostic parameters are listed below. For more information on some of the other parameters you may encounter, please refer to the documentation that ships with RenderMan.

Control (UI)	Default Value	Function
name	PrmanShadingNode	Determines the attribute identifier for this shader node beneath the 'material' attribute. This must be unique among all upstream nodes connected into a single NetworkMaterial node.
nodeType	N/A	Selects the available shader from the dropdown list. The parameters for each shader in the dropdown list are not included, as they are renderer-specific. Use the file browser or your studio's asset management

Control (UI)	Default Value	Function
		browser to select the shader to use.
parameters	N/A	Once you've added a shader, the shader's parameters are populated under the **Parameter** group.
publicInterface		
namePrefix	N/A	Specifies the name's prefix for the exposed parameter.
pagePrefix	N/A	Allows you to organize the shading node's exposed parameters in groups in the NetworkMaterial node's Material Interface.
nameRegExFind	N/A	Finds and deletes the name specified in **namePrefix** field.
nameRegExReplace	N/A	When used with **nameRegExFind**, finds and replaces the name with the name specified by **nameRegExReplace**.
pageRegExFind	N/A	Finds and deletes the name specified in **namePrefix** field.
pageRegExReplace	N/A	When used with **pageRegExFind**, finds and replaces the name with the name specified by **pageRegExReplace**.
PrmanShadingNode parameters continued		
Force Refresh	N/A	Reloads the shader file's information.

Prune

The Prune node removes objects from a scene. Any location that matches the given CEL statement is removed from the output. Any parent location that matches the CEL statement also has all children removed from the output, so there's no need to match all the children if you're pruning out an entire tree of locations. See also Isolate and VisibilityAssign.

Notes:

- To prune out all polymesh objects, use a 'Custom' type statement that looks like this:

  //*{@type=="polymesh"} Change polymesh to whatever type you're interested in to remove that type.
- You don't need to prune out an object to prevent it from being used in a render. The VisiblityAssign node is another way of removing objects from the render without actually removing the object from the scene.

- To see what the Prune is removing, view the node above the prune then click the little arrow on the Prune node's **cel** parameter and select **Find and Select in Scenegraph....**

  After processing for a while, all objects that are to be pruned become selected in the **Scene Graph** tab.

  If nothing is selected, then nothing matches the CEL statement and nothing is pruned.

Connection Type	Connection Name	Function
Input	A	The place in the node graph where you want to remove objects from the scene.

Control (UI)	Default Value	Function
cel	N/A	The CEL statement to use to select locations to remove. The scene graph locations are specified using the Collection Expression Language (CEL). The **cel** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

Rename

This node is useful for renaming scene graph locations according to regular expression matching and substitution. Be aware that many operations are dependent on the names of scene graph locations. Use this with care as it's possible to invalidate subsequent operations by changing scene graph location names.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to renamed scene graph locations.

Control (UI)	Default Value	Function
rootLocation	/root/world/geo	Describes the top-most location on which to perform renaming. The **rename** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
locationTypes	N/A	Accepts a comma-delimited list of scene graph location types on which to act. An empty list acts upon all types.
pattern	N/A	Defines a POSIX-style regular expression on which to match.
replace	N/A	Sets the string replacement. \1 though \9 expand to matched groups in the above pattern. \0 expands to the full match string.

RenderOutputDefine

Specifies output of an image (color, AOV, shadow map, or similar) to a file. In RIB, this means a Display statement.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to define the output settings for rendering.

Control (UI)	Default Value	Function
outputName	primary	Associates a name with the display. Typically primary by default; often shadow for shadow maps, and similar conventions. This name appears in the Render node, along with (or as) the default **primary**.

Control (UI)	Default Value	Function
type	color	Specifies the type of output. • **color** - mostly used to render out rgb beauty files, but also can be used for rendering out z, P(point), N (normals), Ci(final shader color) passes. • **raw** - allows you to directly specify the values for a Display line. Since the output could be anything, Katana doesn't do any colorspace conversion on this output, and can't support tiling. • **script** - run a script on another RenderOutputDefine, like txmake. • **prescript** - run a script before the render is started. • **none** - clears the output. If the output was previously setup by a different RenderOutputDefine node, this removes the entry.
includedByDefault	Yes	When enabled, this Render Definition is sent to the Render node.
rendererSettings		
colorSpace	linear	Sets the output colorspace used.
fileExtension	exr	Sets the output file format.
channel	rgba	Sets the channels to output. You can also set a user-defined channel from a PrmanOutputChannelDefine node.
When fileExtension: exr; convertSettings		
exrCompression	Scanline ZIP	Defines the exr compression method to use. All methods are lossless (with the exception of **Pixar24**, which is lossless but quantizes the pixels to 24-bit float). **Wavelet** is generally preferable as it offers ~2:1 compression even on grainy data. • **None** - • **RLE** - • **Scanline ZIP** -

Control (UI)	Default Value	Function
		• **Block ZIP** - • **Wavelet** - • **Pixar 24** -
exrBitDepth	16	Sets the floating point precision of the rendered exr file: • **16** - half float. This is recommended for all color passes. • **32** - full float. This is recommended for all ncf data arbitrary output variables (AOVs).
exrOptimize	Yes	When enabled, the exr file is written out in an a manner optimized for efficient random tile-access. These optimizations greatly improve memory usage and performance for programs, which process images in tiles.
exrType	Tiled	Sets whether the exr file is written to support: • **Tiled** - random tile access. • **Scanline** - random scanline access.
When fileExtension: exr		
clampOutput	No	When set to **Yes**, post-render clamp negative rgb values to 0, and clamp alpha values to 0-1. **Note:** clampOutput has no effect on NaN and inf values.
colorConvert	Yes	When set to **Yes**, post-render convert rendered image data from linear to the output colorspace specified in the filename. The default value of Yes is suitable for nearly every situation, since the linear output of the render is converted to the colorspace in the filename. A case where you would want to set this to No is if you know the data being rendered is in a colorspace other than linear, such as the re-projection of a log plate, and you want to name the output file log without a linear to log conversion.

Control (UI)	Default Value	Function
When fileExtension: png; convertSettings		
pngBitDepth	16	Sets the bit depth of the rendered file: • **8-bit** • **16-bit**
When fileExtention: rla; convertSettings		
rlaBitDepth	16	Sets the bit depth of the rendered file: • **8-bit** • **10-bit** • **16-bit** • **32-bit**
When fileExtention: tif; convertSettings		
tifBitDepth	16	The bit depth of the rendered file: • **8-bit** • **16-bit** • **32-bit**
tifCompression	LZW	The tif compression method to use: • **None** - No compression method is used. • **LZW** - The LZW compression method is used. This is lossless, so it is usually preferable to use it unless there is an issue with compatibility in the target reader.
When fileExtension: tif		
clampOutput	No	When set to **Yes**, post-render clamp negative rgb values to 0, and clamp alpha values to 0-1. **Note:** clampOutout has no effect on NaN and inf values.
colorConvert	Yes	When set to **Yes**, post-render convert rendered image data from linear to the output colorspace specified in the

Control (UI)	Default Value	Function
		filename. The default value of **Yes** is suitable for nearly every situation, since the linear output of the render is converted to the colorspace in the filename. A case where you would want to set this to **No** is if you know the data being rendered is in a colorspace other than linear, such as the re-projection of a log plate, and you want to name the output file log without a linear to log conversion.
When fileExtension: jpg		
jpgQuality	100	The quality to use when generating the jpg file. Higher values generate larger file sizes, with 100 representing the best quality image and 0 representing the lowest.
rendererSettings parameters continued		
computeStats	None	Allows you to compute image statistics as a post process, appending as exr metadata. Select: • None • Raw • Depth **Note:** In depth mode, zero values and very large values are ignored. In both modes, only the region within the dataWindow is considered.
tempRenderLocation	N/A	
cameraName	N/A	Describes the scene graph location of camera to render from. If empty, render from the camera specified in **renderSettings.cameraName** at **/root**. The **cameraName** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location

Control (UI)	Default Value	Function
		Widget Type in Common Parameter Widgets.
locationType	local	
When locationType: file; locationSettings		
renderLocation		Specify the render location, or bring up the file browser or your studio's asset management browser to select the location to use. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.



Note: RenderOutputLocation plug-ins that are shipped as source and can be found in `plugins/Src/RenderOutputLocations`.

ReverseNormals

ReverseNormals reverses any point and vertex normals on locations matching its CEL parameter. Point normals are represented by a **geometry.point.N** attribute, and vertex normals are represented by a **geometry.vertex.N** attribute. If neither of these attributes are present, the node has no effect. If they are both present, they are both reversed. Any other normal attributes, such as surface normals, are not recognized or modified by the node.

Connection Type	Connection Name	Function
Input	A	The place in the node graph where the point and vertex normals on specified locations are reversed.

Control (UI)	Default Value	Function
celSelection	N/A	Sets the normals location to be reversed.

Control (UI)	Default Value	Function
		The scene graph locations are specified using the Collection Expression Language (CEL). The **celSelection** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.

ShadingGroup

The ShadingGroup node is designed to keep your workspace inside a NetworkMaterialCreate node organized by allowing you to group sections of your shading node network together. Inside a ShadingGroup node, there are fixed input and output bars, which are used to connect the nodes within the group to the rest of the network.

Node Parameters

The **Node Parameters** tab is empty until a shading node network exists inside the ShadingGroup node. Once the ShadingGroup node contains a shading node network, the parameters of the nodes which are in use are automatically promoted and displayed within the **Node Parameters** tab.



Note: The parameters are only promoted if the parameter is editable from within the node **Parameters** itself.

The promoted parameters are organized by node, page and parameter. This makes it easy to make adjustments to the parameters in use without having to enter the ShadingGroup.



Note: For more information on the ShadingGroup node, see Organizing Shading Networks with ShadingGroup Nodes documentation.

Material Interface

Control (UI)	Default Value	Function
Name	N/A	A list displaying all promoted parameters, organized in the same way as they were grouped when promoted.
Source	N/A	The path to each different parameter.



Note: For more information on the uses of the Material Interface, see Node Parameters and Interface Controls documentation.

ShadingNodeArrayConnector

The ShadingNodeArrayConnector allows you to collect the connected shading nodes' outputs and build an input connection for shading node array parameters. The ShadingNodeArrayConnector node only works with shading nodes, which define the parameter as an array.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	Connect shading node array parameters to the various inputs.

ShadingNodeSubnet

The ShadingNodeSubnet node allows you to group shading nodes together for a better organization of your shading network. You can then connect the ShadingNodeSubnet node to the NetworkMaterial node. To do the inverse (explode the group of shading nodes), select the ShadingNodeSubnet node and press **U**.

Control (UI)	Default Value	Function
Subnet Material Interface or right-click interface table		
Remove	N/A	Removes the selected shading node parameter from the Subnet Material Interface.
Refresh	N/A	Refreshes the Subnet Material Interface.
Add Exposed Parameters	N/A	Adds the exposed parameters of all of the shading nodes contained within this ShadingNodeSubnet to the Subnet Material Interface.

Transform3D

Adds transform attributes to scene graph locations allowing you to control 3D objects in the Viewer.

Note: Manipulates the xform attribute and is used by the AttributeEditor node.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to add transform attributes to scene graph locations.

Control (UI)	Default Value	Function
path	/root/world/geo	Sets the path to a scene graph location. The **path** parameter options are available by clicking the dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
order	Scale Rot Trans	Sets the order to apply the transform.

Control (UI)	Default Value	Function
rotateOrder	Rx Ry Rz	Sets the order of each rotation.
stackOrder	First	Sets whether to apply before or after the transforms.
translate	0.0, 0.0, 0.0	Moves the object up, down, left, right, in or out (of 3D space).
rotate	0.0, 0.0, 0.0	Specifies the pivoting around the pivot (axis).
scale	1.0, 1.0, 1.0	Sets the scale (on individual axis of x, y or z).
pivot	0.0, 0.0, 0.0	Sets the point around which the translate and rotate happens.
uniformScale	1	Scales the translate, rotate and scale uniformly.
makeInteractive	No	When set to **Yes**, you can drag objects in the Viewer and Katana retains the information from the Viewer.
adjustParentBounds	Yes	Specifies whether or not to adjusts the bound attribute of the parent locations affected by the transformations of the child.

TransformEdit

The TransformEdit node allows you to make changes to the transform attributes of a scene graph location.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to transform the attributes of the specific scene graph location.

Control (UI)	Default Value	Function
path	/root/world/geo	Sets the path to a scene graph location. The **path**

Control (UI)	Default Value	Function
		dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.
action	replace global transform	Determines the node's behavior: • **override interactive transform** - overrides the values set in the scene graph location. • **append new transform** - adds values to the existing values set in the scene graph location. • **replace global transform** - replaces the global transform attributes relative to the origin of the world.
rotationOrder	XYZ	Sets the order in which the rotation is applied: **XYZ**, **XZY**, **YXZ**, **YZX**, **ZXY**, **ZYX**.
When action: append new transform		
stackOrder	First	In Katana the local transform at any location is created using a stack of transforms. The **stackOrder** parameter specifies whether the new transform is appended at the first or last position in this stack.

VelocityApply

Creates extra time samples on the P or Pw attribute of a shape using the V or v attribute describing velocity in units per second.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to create time samples.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
velocityAttribute	N/A	The name of the attribute representing the velocity information to be used by the node. If the parameter is not set, the following attributes are checked: • **geometry.point.V** • **geometry.point.v** • **geometry.arbitrary.v**
velocityUnits	units / second	Units to be used to interpret the values stored in the velocity attribute, with the following options: • **units / second** • **units / frame**
velocityScale	1	Defines a multiplier on the velocity attribute, where 1 = no change.
fps	24	Defines frames per second. Used to determine the amount of the velocity, which is defined in units per second, to apply to the geometry attributes.

ZoomToRect

The ZoomToRect node zooms and/or crops the render by setting crop window and screen window attributes. When this node is edited, a guide rectangle is drawn in the **Monitor** tab. The guide can be resized to adjust what portion of the render is zoomed and/or cropped.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where the render is cropped to specific window and screen window attributes.

Control (UI)	Default Value	Function
rect		
<resolution>	Dependent on Project Settings	List of preset sizes to zoom to.
rect >	N/A	For more information, refer to the Rectangle Widget Type in the Common Parameter Widgets.
left	0	Sets the left position of the rectangle.
bottom	0	Sets the bottom position of the rectangle.
width	2048	Sets the width of the rectangle.
height	1556	Sets the height of the rectangle.
ZoomToRect parameters continued		
zoom	Yes	When **Yes**, sets what portion of the render to zoom in to.
crop	Yes	When **Yes**, sets what portion of the render to crop out.

USD Nodes

KatanaToUsd

The KatanaToUsd node converts Katana's geolib data to USD data, enabling you to handle multiple data types and complete the USD "round-trip" (alongside the UsdToKatana node). For more information and

examples, see Conversion Nodes - KatanaToUsd/ UsdToKatana.

This node will convert the following data to USD:

Primitive Types: If the location has no transform attribute, it is defined as a Scope Prim. There is an option to define as Xform Prim - see table below.

Geometry: the KatanaToUsd node converts any geometry to USD data as well as subdivided meshes.

Cameras: Both perspective and orthographic cameras can be converted. However, attributes that are not set when creating a camera in Geolib, such as focal length, are set to a default value and saved to info.usd.

Materials: Fully fledged material conversion, including: single Material nodes, NetworkMaterialCreate and NetworkMaterialEdit nodes, child materials, material assignments, layout attributes, parameter interface attributes, default values, material overrides and its round-tripping, deferred material ops as well as renderer specific materials. An option can be found which disables the conversion of material assignment if the target primitive is being ignored - see table below.

Lights: Currently, Katana lights get converted to its USD counterpart as long as a matching shader identifier can be found in the shader registry. There is an option to convert light list information - see table below.

Face sets and Time samples: Face sets are converted over to its USD equivalent; UsdGeomSubsets. KatanaToUsd supports the conversion of time sampled data, such as multi-sampled attributes and time sampled attributes.

Custom Attributes: KatanaToUsd allows for generic conversion of Katana attributes to USD attributes. More detailed information on this as well as the attribute convention can be found in the developer guide.

Inputs

Connection Type	Connection Name	Function
input	in	The place in the node graph for the conversion to USD data to take place.

Controls

Control (UI)	Default Value	Function
location	N/A	The location path specifying the hierarchy to convert to USD. For example, /root/world/geo will convert this part of the scene, along with all paths below.
prefixPrimSpec	over	Determines the specifier of the Prim path defined in the prefixPath.
prefixPath	N/A	The parent hierarchy for the converted prims. This determines where the converted information should go in Usd's section of the Scene Explorer.
pruneLocations	enabled	Determines whether to remove the converted locations from Katana's side of the Scene Explorer.
strictMode	disabled	Determines whether the conversion stops when an attribute conversion fails. If enabled, an FnError prim will be generated on first error and conversion will be stopped. If disabled (default), conversion continues despite failures
incomingFocalLengthUnit	mm	Unit of the focal length to convert. Either millimeters or centimetres. Set to millimeters by default.
incomingApertureUnit	cm	Unit of the aperture to convert. Either millimeters or centimetres. Set to centimetres by default.
disableMeshConversion	disabled	Disables converting the mesh and transform data on polymesh or submesh locations. This will not change which locations are pruned from the Katana Scene. This is useful if only the material assignments or other mesh overrides are desired in the conversion.
lightListLocation	N/A	Location path to set where the light list should be written, If none is specified, no light list information is written.
relativeLightCompositionArc	inherits	The composition arc for the Prim containing a light deriving fromMeshLightAPI or VolumeLightAPI, to compose the attributes from the Prim storing the MeshLightAPI or VolumeLightAPI onto the Mesh/Volume itself.
onlyConvertIfMaterialExists	enabled	If set, any material assignment properties targeted to

Control (UI)	Default Value	Function
		materials that are not converted in the KatanaToUsd node, will be removed from the primitive.
groupsAsXforms	disabled	When disabled, they will be defined as Scope prims, unless the locations contain transformation attributes, in which case they will be converted as Xform prims.If enabled, all Group location types will be defined as Xform type prims.



Note: You can find an **ignore** Path widget on the KatanaToUsd node. This widget is used to select certain Katana locations to be ignored by the KatanaToUsd node and won't be converted over to the USD native space.



Note: The KatanaToUsd node currently supports the **primtypes** mentioned above as well as **custom primdecorators**.

Tip: Use the KatanaToUsd node in conjunction with the Native USD Layer Export workflow when handling a lot of upstream data to improve performance.

Tip: See Conversion Nodes - KatanaToUsd/ UsdToKatana examples to learn how to convert between Geolib and USD data.

UsdActiveSet

UsdActiveSet allows you to activate or deactivate any number of prims in the stage. Deactivation is useful when you want to remove prims from the scene graph, and exclude them from stage composition. Due to its focus on non-destructive editing, USD does not support direct prim deletion. Instead, deactivation allows you to hide the prim and its descendents from scene graph computation.



Note: Activation and deactivation of prims is available within the Scene Explorer using Working Sets. Working Sets provide a quick shortcut for testing late changes with previews in the Viewer or Live Render. They act as a temporary override and have no effect on the final render. However, using a node to deactivate a prim expresses an opinion that the prim should be deactivated for the final render.

Connection Type	Connection Name	Function
Input	in	The incoming scene graph data that the node will operate on or modify.

Control (UI)	Default Value	Function
primPaths	N/A	The path at which to set the active or inactive prim data. If you set this to the root path ("/") then the status will apply to to all root prims.
active	off	Uncheck to deactivate and check to activate.

UsdAttributeSet

In USD, the attributes are essentially the defining properties of the prim. As a simple example, one of the properties of cube prim is "size", which could appear as follows:

```
#usda 1.0
def Cube "myCube"
{
 float size = 2.0
}
```

An attribute has a data type (the type is "double" in the example above), name, and value. The value may be numeric, a string, or an array. Unlike metadata, the value of an attribute may be time-sampled, in that the attribute may have several values, each of which is active at a given timecode or frame.

Attributes are one of the two defining properties of a prim, together with relationships.



Note: For more details see Attribute in OpenUSD's USD Glossary.

Setting and Blocking Attributes

When assigning an attribute to a prim, you can choose between one of three actions:

- **Create/Override** - Add a new attribute or override the value of an existing one.
- **Block** - Disable the attribute, effectively blocking its effect on the prim.
- **Force Default** - Force the attribute back to its default value.

As **Block** and **Force Default** don't require a value, you need only supply the **attributeName**.

Create/Override requires all the attribute settings including **attributeName**, **attributeType**, and **value**. You also need to set **isCustom** to indicate if the attribute is pre-defined or a new, custom attribute.

For time-sampled values, set **multisample** to **Yes**. For example, say you're animating a cube to grow in size, the attribute in USD code, could look like this:

```
def Cube "GrowingCube"
{
 float size.timeSamples = {
  0: 1.0,
  2: 2.0,
  3: 4.0,
 }
 }
```


Tip: You can drag and drop an attribute from the Attributes Tab into the UsdAttributeSet node's Parameters to have it adopt the attributeName, attributeType, and value.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	N/A	The paths to the Prims that you want to assign this attribute to.
action		Choose how you want to apply the attribute update. • **Create/Override** - Create the attribute, or if it exists already, assign the new value. • **Block** - Disable the attribute assignment in the prims, removing their effect from stage composition. • **Force Default** - Force the attribute to adopt its default value.
attributeName	N/A	Assign a name to the attribute.
attributeType	Double	Choose from one of the available data types for the attribute.
multisample	Yes	Select **Yes** if the data is time-sampled. Select **No** if the value should not vary over time.
isCustom	No	Select **Yes** if this is a custom attribute.
value	N/A	Assign a value to the attribute.

UsdCamera

This node allows you to create native USD cameras in Katana. See more on Creating Native USD Cameras.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPath	N/A	Location Paths of where to create the camera. For example, **/cameras/camera**.
primSpec	define	Choose the specifier for the prim. This is the role or function of the prim in the scene description. • **define** - defines a new, complete specification for a prim and its properties within a layer. • **class** - defines a prim meant to act as a template for other prims. It ensures uniformity and reusability throughout the scene by allowing other prims to adopt or specialize it, inheriting its characteristics. • **over** - indicates that the prim is an override. It's used to non-destructively modify prims defined in lower-priority layers. A prim with an "over" specifier is meant to modify or extend the properties or hierarchy of existing prims, without redefining them entirely.
primSpecHierarchy	disabled	Determines whether to create all prims in the provided hierarchy with the primSpec. If disabled, all but the final Prim will take on the primSpec, to avoid writing over primSpecs in

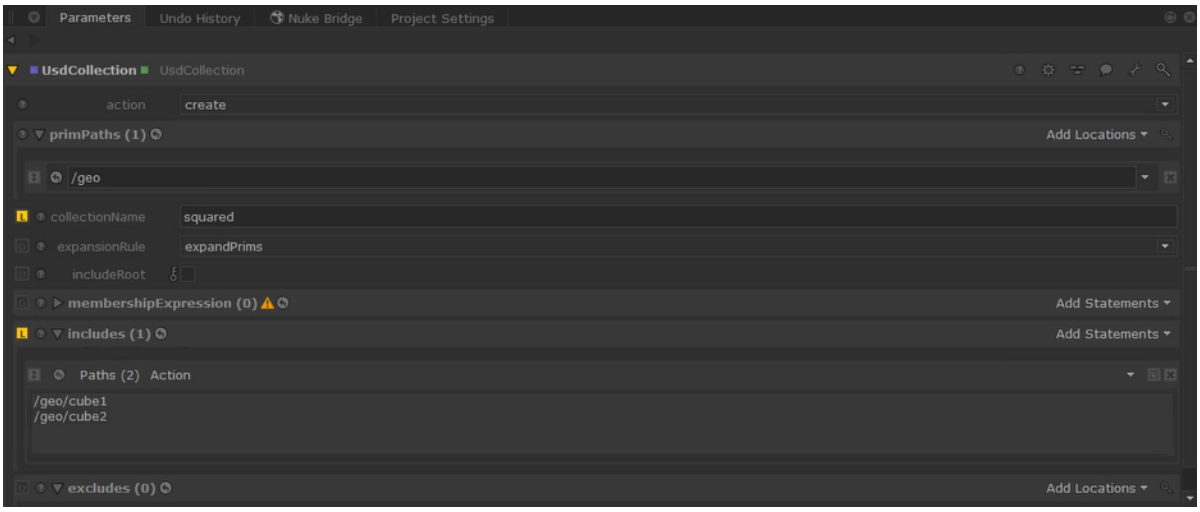
Control (UI)	Default Value	Function
		lower order layers.
kind	assembly	Determines what Kind metadata should be set on the prims.
properties > clippingPlanes		
clippingRange	1.0	Near and far clipping distances in scene units.
exposure	0	Exposure adjustment, as a log base-2 value.
focalLength	50	Perspective focal length in tenths of a scene unit.
focusDistance	0	Distance from the camera to the focus plane in scene units
fStop	0	Lens aperture.
horizontalAperture	20.955	Horizontal aperture in tenths of a scene unit.
horizontalApertureOffset	0	Horizontal aperture offset in the same units as horizontalAperture.
projection	perspective	Projection type, either “perspective” or “orthographic”.
purpose	default	Purpose is a classification of geometry into categories that can each be independently included or excluded from traversals of prims on a stage, such as rendering or bounding-box computation traversals.
properties > shutter		
stereoRole	mono	Left or right camera of a stereo setup or mono.
verticalAperture	20.955	Vertical aperture in tenths of a scene unit.
verticalApertureOffset	0	Vertical aperture offset in the same units as verticalAperture.
visibility	inherited	Visibility is meant to be the simplest form of "pruning" visibility that is supported by most DCC apps. Visibility is animatable, allowing a sub-tree of geometry to be present for some segment of a shot, and absent from others; unlike the action of deactivating geometry prims, invisible geometry is still available for inspection, for positioning, for defining

Control (UI)	Default Value	Function
		volumes, etc.
Properties > transform		
makeInteractive	Yes	When set to "Yes", the selected path can be interactively manipulated in the Viewer. This option is only available when the mode parameter is set to "Single Path".
xformOpName	UsdCamera	The name to be given to the XformOp when stored in USD.
order	Scale Rot Trans	Select the order by which the three transformations are applied.
rotateOrder	Rx Ry Rz	Select the axes order by which the rotation takes place. By default the rotation is first around the x axis, then y, and finally z.
translate	0.0	The amounts to translate along the x, y, and z axes.
rotate	0.0	The amounts to rotate the prims around the x, y, and z axes.
scale	1.0	The amounts to scale the prims along the x, y, and z axes.
pivotTranslate	0.0	The amounts by which the pivot point for transformations is translated along the x, y, and z axes.

UsdCollection

Pattern-Based Collections (PBC) was introduced in USD 23.11 and is now fully integrated in Katana, with the addition of the UsdCollection node. This allows you to define and modify USD collections.

The UsdCollection node has two modes that are found in the **action** parameter's dropdown: **create** and **edit**.



action > create

This mode defines a USD collection.

- Add a location in the **primPath** parameter and give your collection a name in the **collectionName** parameter.
- Use **membershipExpression**, **include** and **exclude** widgets to add paths.

action > edit

This mode allows modifying incoming collections.

- Specify a location in the **primPath** parameter and select the collection within the **collectionName** parameter.
- Inputs will then be automatically pre-populated with the paths set within the collection, and can be further edited.

Note: For further information about PBC, see Pattern-Based Collections with USD.

Tip: You can see a list of the CEL and USD (PBC) Collections in your project in the Collections panel at the top of the Scene Explorer and via the Collection working set indicators. See Collections in the Scene Explorer.



Warning: If your collection specifies includes/excludes (relationship mode) and a membership expression (expression mode), then the includes/excludes take priority and the membership expression will be ignored. For this reason, relationship mode collections set by includes/excludes will not be accepted by PBC widgets.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

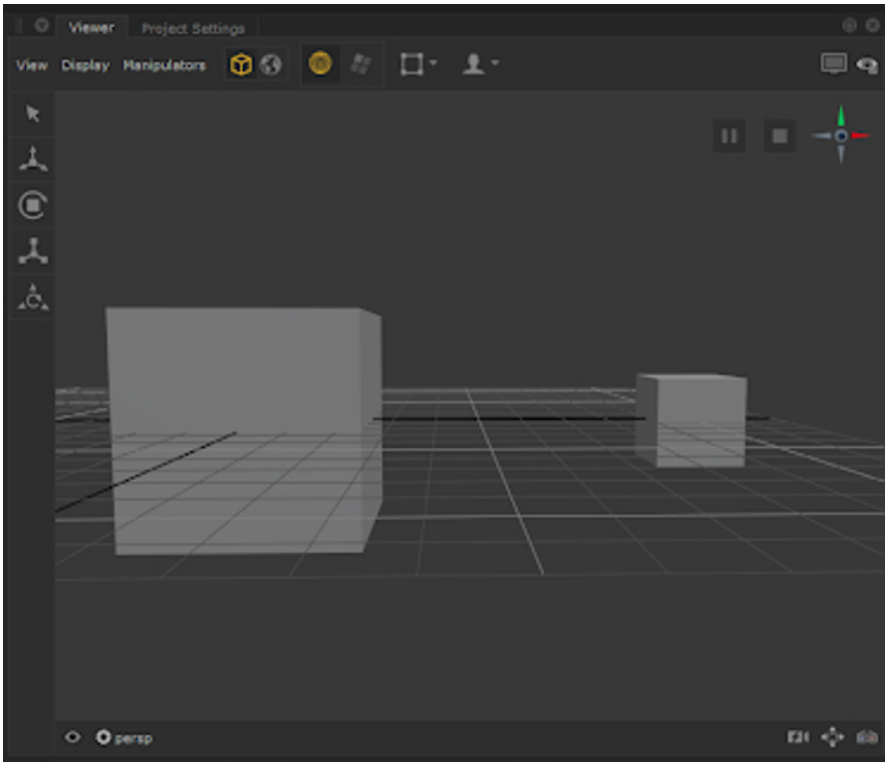
Control (UI)	Default Value	Function
action	create	The action of this node is to either create a new or edit an existing collection. If editing an existing collection, only one prim path can be set, otherwise when creating a collection, collections can be created at one or more prim paths.
primPaths	N/A	The prim paths on which to create the collection
collectionName	N/A	The name of the collection to create or edit.
expansionRule	expandPrims	Specifies how the paths that are included in the collection must be expanded to determine its members. **explicitOnly** - only paths in the includes section and not in the excludes section belong to the collection. **expandPrims** - all the prims at or below the relationship includes (and not under the relationship excludes) belong to the collection. Any property paths included in the collection would also be honored.

Control (UI)	Default Value	Function
		This is the default behavior as it satisfies most use cases. **expandPrimsAndProperties** - like expandPrims, but also includes all properties on all matched prims. We're still not quite sure what the use cases are for this, but you can use it to capture a large number of UsdObjects very concisely.
includeRoot	disabled	Boolean attribute indicating whether the pseudo-root path should be counted as one of the included target paths. This separate attribute is required because relationships cannot directly target the root.

UsdInheritSet

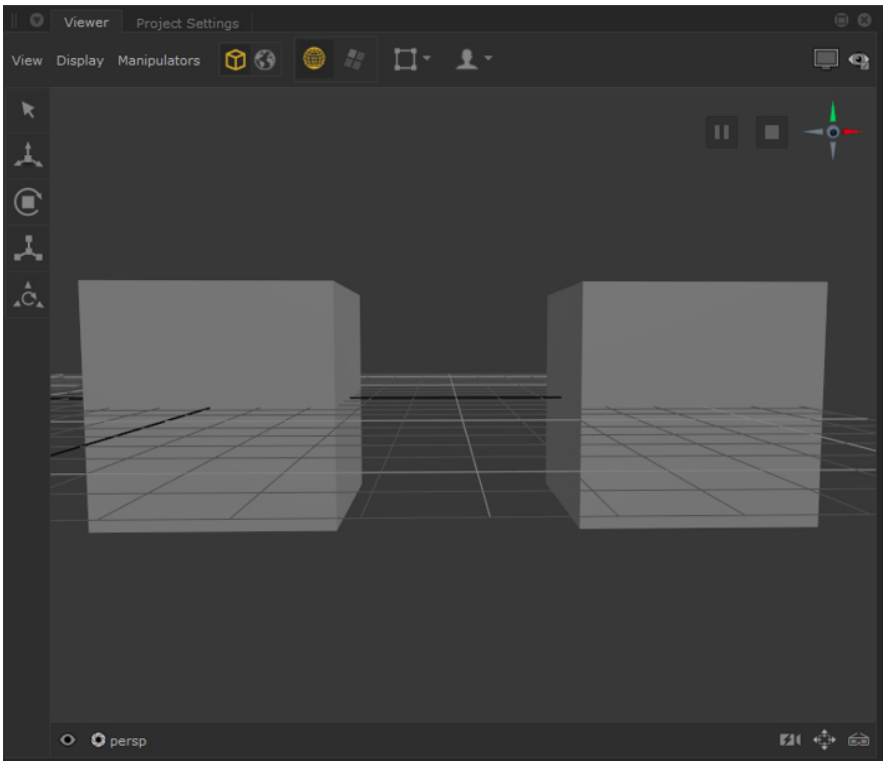
This node allows you to inherit the hierarchy and properties from one source prim to other prims. If you make updates to the source prim, all the prims inheriting from it will automatically reflect those updates. This is particularly useful when dealing with multiple instances of a prim, as it ensures that you only need to make updates once, and those changes will be applied to all instances.

For example, imagine you have two cubes, cube1, and cube2, which you've created using two UsdPrimCreates. Cube1 is larger than cube2.



To make cube2 inherit the properties of cube1:

1. Create a **UsdInheritSet** and open the **Parameters tab**
2. Set **primPaths** to /cube2
3. Set the **inheritPath** to /cube1
4. In this case, **listPosition** can be set to either its default **prepend** or **append** since the order doesn't affect the cube inheriting another cube's size. This setting becomes more important when there are multiple items on the list.



Cube 2 now inherits the properties of cube 1.

Input

Connection Type	Connection Name	Function
Input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	n/a	Specify the locations of the prims where you want to add an

Control (UI)	Default Value	Function
		inheritance, such as /geo. The inheritance is set on each specified prim.
inheritPath	n/a	Specify the location of the prim you wish to inherit from.
listPosition	prepend	Specifies how Inherits are added to the scene. Options are: • **append** - adds new inherits by placing them at the end of the list. This ensures that they are evaluated after existing values, giving them the potential to override preceding definitions. • **prepend** - adds new inherits to the front of the list, causing them to be evaluated before, and potentially overridden by values added later or in subsequent layers. • **delete** - removes the inherits from the list, preventing their properties from influencing the composed prim in the scene. • **reset to explicit** - clears any existing inherits on the prim and sets the inherit list to be only the ones you're adding. It essentially instructs the USD scene to ignore any inherits on the prim(s) and only use the new ones. For more information, refer to the USD Glossary under List Editing.

UsdKindSet

The UsdKindSet node enables you to modify the **Kind** of a prim. In USD, Kind is the organizational role of an element in the hierarchy. It is categorized as values such as **component** for leaf level items or **assembly** for a collection of components. Other options include **group**, **model**, and **subcomponent**.

See Kind in OpenUSD's USD Glossary.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
mode	Paths	The prims may be specified by either **Paths** or **Pattern Based Collection**.
primPaths	N/A	The path at which to set the Kind prim metadata.
kind	assembly	Determines what Kind metadata should be set on the prims, to determine the organizational role of the prim. • **assembly** - for a collection of components • **component** - for leaf level items • **group** - for groups of models • **model** - the kind base value, which is abstract • **subcomponent** - for sub sections of components

UsdLayerDefine

This node is used in conjunction with UsdLayerExport in order to define which changes to capture in the layer export. **UsdLayerDefine** acts as a start point for the exported layer and is only required if you are exporting from a point where nodes exist above. See Native USD Layer Export to learn more about the workflow.

Inputs

Connection Type	Connection Name	Function
input	in	The place in the node graph where you want to start an export to USD.

Controls

Control (UI)	Connection Name	Function
asset	N/A	The asset which the exported layer edits will be added to.

UsdLayerExport

This node exports Native USD data to a USD layer. See Native USD Layer Export to learn more about the workflow.

Inputs

Connection Type	Connection Name	Function
input	in	The place in the node graph where you want to export to the USD layer.
define	define	Determines the start point for the exported layer if you have used a UsdLayerDefine node.

Controls

Control (UI)	Connection Name	Function
saveTo	N/A	The asset path to write the exported layer to.
usdFileFormat	usd	The type of USD file to write. This will override any extension that might be set in the **saveTo** parameter and update `fileExtension` in the asset API.
useExported	disabled	When enabled, the content of this node will be bypassed and will instead utilize the USD file that was last exported from this node.
timeSamples	Current Frame	If **Current Frame** is selected, only the current frame set on the timeline will be exported. If **Frame Range** is selected, the frame range set in the **Parameters** tab will be exported. If **Project Settings** is selected, the frame range set on the **inTime** and **outTime** parameters in the Project Settings tab will be exported. If **Defined Layer Metadata** is selected, the frame range set on the `startTimeCode` and `endTimeCode` metadata in the layer set on an upstream UsdLayerDefine will be exported.
framesRange	0.0, 0.0	The range of frames which will be exported to the layer. The end frame is inclusive. This is only used when **timeSamples** is set to **Frame Range**.
samplesPerFrame	1	Sets the number of time samples to export per frame.
exportMethod	Keep Composition Arcs	If **Keep Composition Arcs** is selected, SubLayers are kept if explicitly added via Composition Arc node, as well as any

Control (UI)	Connection Name	Function
		References. If **Flatten Sublayers, Keep References** is selected, SubLayers are flattened together while any other composition arcs are kept. If **Flatten All** is selected, all SubLayers and References are flattened into a single layer.
removeDuplicateTimeSamples	enabled	Runs a process prior to export to deduplicate all attributes within the scope of the layer being exported. For example this means that it can remove duplicates within sublayers if the exportMethod option was to **Flatten Sublayers, Keep Reference**s.
makeSingleTimeSamplesStatic	enabled	Whilst removing duplicate time samples, attributes with only a single time sample will be exported as a static (non-sampled) value.

UsdLayerExportGroup

This node allows you to select multiple nodes and add them to the Group ready for export without having to do the setup manually. To do this, select the nodes and go to **Edit** > **Collapse Selected Nodes to UsdLayerExportGroup**.

SeeNative USD Layer Export to learn more about the workflow.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on.

Controls

Control (UI)	Connection Name	Function
asset	N/A	The asset which the exported layer edits will be added into.
saveTo	N/A	The asset path to write the exported layer to.
usdFileFormat	usd	The type of USD file to write. This will override any extension that might be set in the **saveTo** parameter and update `fileExtension` in the asset API.
useExported	disabled	When enabled, the content of this node will be bypassed and will instead utilise the USD file that was last exported from this node.
timeSamples	Current Frame	If **Current Frame** is selected, only the current frame set on the timeline will be exported. If **Frame Range** is selected, the frame range set in the **Parameters** tab will be exported. If **Project Settings** is selected, the frame range set on the **inTime** and **outTime** parameters in the Project Settings tab will be exported. If **Defined Layer Metadata** is selected, the frame range set on the `startTimeCode` and `endTimeCode` metadata in the layer set on an upstream UsdLayerDefine will be exported.
framesRange	0.0, 0.0	The range of frames which will be exported to the layer. The end frame is inclusive. This is only used when **timeSamples** is set to **Frame Range**.
samplesPerFrame	1	Sets the number of time samples to export per frame.
exportMethod	Keep Composition	If **Keep Composition Arcs** is selected,

Control (UI)	Connection Name	Function
	Arcs	SubLayers are kept if explicitly added via Composition Arc node, as well as any References. If **Flatten Sublayers, Keep References** is selected, SubLayers are flattened together while any other composition arcs are kept. If **Flatten All** is selected, all SubLayers and References are flattened into a single layer.
removeDuplicateTimeSamples	enabled	Runs a process prior to export to deduplicate all attributes within the scope of the layer being exported. For example this means that it can remove duplicates within sublayers if the exportMethod option was to **Flatten Sublayers, Keep Reference**s.
makeSingleTimeSamplesStatic	enabled	Whilst removing duplicate time samples, attributes with only a single time sample will be exported as a static (non-sampled) value.

UsdLayerWrite

UsdLayerWrite creates a USD layer, allowing you to write USD data directly to a stage from an editable scene description. This node is a useful tool for quickly adding content to specific locations within your scene.

You can edit the USD file within the **Parameters** tab or use an external text editor.



Note: UsdLayerWrite primarily serves as a tool for debugging your pipeline, providing a quick and straightforward means of testing and validating your USD setup.

Control (UI)	Default Value	Function
layerContent	#usda 1.0 (defaultPrim = "root")	The scene description using standard USDA.

Control (UI)	Default Value	Function

	def Xform "root"
	{ def Xform "world"
	{
	}
	}	

Edit in External Editor	N/A	Click this to edit the scene description using a text editor outside Katana. The choice of editor is set up in **Edit** > **Preferences** > **externalTools** > **Editor**. Under editor enter the location of the editor application. Katana monitors the text files you edit and updates the node when changes are detected.

UsdLight

This node allows you to create and edit **native USD lights** in Katana. See more about the workflow and examples on the Native USD Lights page.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Connection Name	Function
action	create	Determines whether to either create a new light prim or edit an existing light prim.
primPath	/lights/light	Location Paths of where to create or edit Prim. For example, **/lights/light**.
primSpec	define	Choose the specifier for the prim. This is the role or function of the prim in the scene description. • define - defines a new, complete specification for a prim and its properties within a layer. • class - defines a prim meant to act as a template for other prims. It ensures uniformity and reusability throughout the scene by allowing other prims to adopt or specialize it, inheriting its characteristics. • over - indicates that the prim is an override. It's used to non-destructively modify prims defined in lower-priority layers. A prim with an "over" specifier is meant to modify or extend the properties or hierarchy of existing prims, without redefining them entirely. For more information, refer to the USD Glossary under Specifier.
lightType	N/A	Determines the type of light this prim will be. When in edit mode, this value is determined by the incoming light prim.
primSpecHierarchy	enabled	Determines whether to create all prims in the provided hierarchy with the primSpec. If disabled, all but the final Prim will take on the primSpec to avoid writing over primSpecs in lower order layers.
blockUpstreamAttributes	disabled	Determines whether to block all Attributes

Control (UI)	Connection Name	Function
		provided by the light Prim type.
Properties > inputs		
color	1.0000	The color of the light (rgba).
colorTemperature	6500	The color temperature of the light.
diffuse	1	The amount of diffusal of the light.
enableColorTemperature	disabled	Enable or disable color temperature.
exposure	0	The amount of exposure of the light.
intensity	1	The amount of intensity of the light.
length	1	The length of the light.
normalize	disabled	Normalizes the light's power with the shape of the light.
radius	0.5	The radius of the light.
Properties > inputs > shadow		
color	0.0000	The color of the shadow.
distance	-1	The distance of the shadow.
enable	enabled	Enable or disable the shadow.
falloff	-1	The falloff of the shadow.
falloffGamma	1	The gamma of the shadow's falloff.
Properties > inputs > shaping		
specular	1	The amount of specular shaping.
Properties > inputs > shaping > cone		
angle	90	The angle of the cone in degrees.
softness	0	The softness of the cone.

Control (UI)	Connection Name	Function
focus	0	The focus of the cone.
focusTint	0.0000	The color of the cone's focus tint.
Properties > inputs > shaping > ies		
angleScale	0	The scale which defines the angular distribution of the light.
file	N/A	A file for an IES light which describes the angular distribution of the light.
normalize	disabled	Normalizes the IES light's power with the shape of the light.
Properties > inputs > texture		
file	N/A	This field is available for light types that require a texture, such as the DomeLight and Rectlight
format	automatic	Format for the incoming file.
Properties > inputs > light > filters		
materialSyncMode	noMaterialResponse	For a LightAPI applied to geometry that has a bound Material, which is entirely or partly emissive, this specifies the relationship of the Material response to the lighting response. Choose from **noMaterialResponse**, **materialGlowTintsLight**, and **independent**.
shaderId	Same as lightType	Default ID for the light's shader. This defines the shader ID for this light when a render context specific shader ID is not available.
Properties > inputs > proxyPrim		
purpose	default	Purpose is a classification of geometry into categories that can each be independently included or excluded from traversals of prims on a stage, such as rendering or bounding-box computation traversals. See for more detail

Control (UI)	Connection Name	Function
		about how purpose is computed and used.
treatAsLine	disabled	A hint that this light can be treated as a 'line' light (effectively, a zero-radius cylinder) by renderers that benefit from non-area lighting. Renderers that only support area lights can disregard this.
visibility	inherited	Visibility is meant to be the simplest form of "pruning" visibility that is supported by most DCC apps. Visibility is animatable, allowing a sub-tree of geometry to be present for some segment of a shot, and absent from others; unlike the action of deactivating geometry prims, invisible geometry is still available for inspection, for positioning, for defining volumes, etc.
Properties > transform		
xformOpName	N/A	The name to be given to the XformOp when stored in USD.
order	Scale Rot Trans	Select the order by which the three transformations are applied.
rotateOrder	Rx Ry Rz	Select the axes order by which the rotation takes place. By default the rotation is first around the x axis, then y, and finally z.
translate	0.0	The amounts to translate along the x, y, and z axes.
rotate	0.0	The amounts to rotate the prims around the x, y, and z axes.
scale	1.0	The amounts to scale the prims along the x, y, and z axes.
pivotTranslate	0.0	The amounts by which the pivot point for transformations is translated along the x, y, and

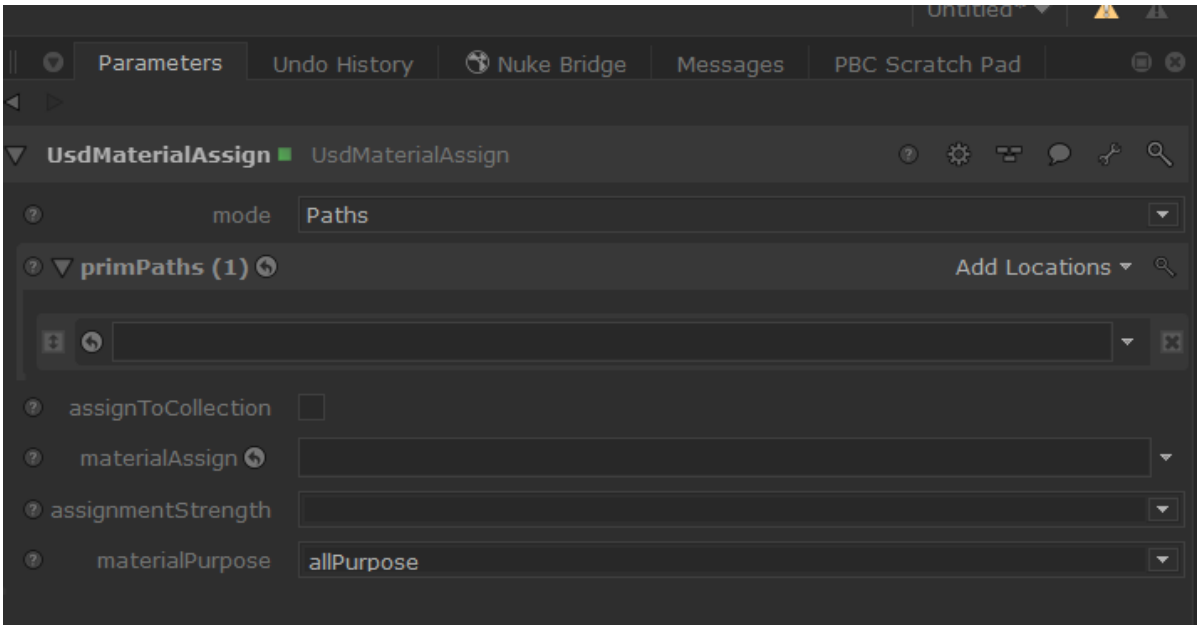
Control (UI)	Connection Name	Function
		z axes.
makeInteractive	Yes	When set to "Yes", the selected path can be interactively manipulated in the Viewer. This option is only available when the mode parameter is set to "Single Path".

Note: This section will show different options depending on the type of light you select.

UsdMaterialAssign

The UsdMaterialAssign node assigns USD materials to geometry, either using Paths or PBC expressions. Its **assignmentStrength** and **materialPurpose** can also be set.

To see more about UsdMaterialAssign in practice, see UsdMaterialAssign Workflows.



Inputs

Connection Type	Connection Name	Function
input	in	The material in the node graph that you want to apply to the geometry in the scene graph.

Controls

Control (UI)	Default Value	Function
mode	Paths	The prims to add the assignment to may be specified either using a pattern or paths.
primPaths	N/A	The paths to the Prims you wish to assign the Material to.
assignToCollection	disabled	Determines whether to append a material to a collection or not.
materialAssign	N/A	The path to the Material you wish to assign the Prims to.
assignmentStrength	N/A	Determines the assignment strength (fallbackStrength, weakerThanDescendants, strongerThanDescendants). The default value is fallbackStrength
materialPurpose	allPurpose	Specifies the material purpose. If set to a value other than the default allPurpose, the assignment applies only to the specified material purpose. Can be set to any string value, current options show the available USDShadeTokens.

UsdMetadataSet

Metadata provides a useful addition to the data stored for layers, prims, and attributes. The metadata is supplementary to core data items and may have a role defined by the schema or may be entirely custom

according to the needs of the studio.

Metadata could simply provide descriptive information or documentation for attributes in the prim. Alternatively, It may contain information for custom workflows or tools. For example, renderer-specific metadata could provide hints about lighting for the studio's choice of renderer.

Metadata is different from most other data in that it cannot be time-varying according to timecode or frame count. Once set, metadata remains fixed for the duration of the scene.

New USD metadata is declared using the 'custom' keyword preceding the data type, metadata name, and value, for example:

```
custom string lightingArtist = "JaneJones"
```

Setting USD Metadata

UsdMetadataSet lets you set metadata at one of three levels, selected using the **hierarchy** dropdown:

- **Attribute** - set metadata on prim attributes contained within selected prims
- **Layer** - set metadata on the top-level pseudo root
- **Prim** - set metadata on the selected prims

For **Atrribute** and **Prim** you need to populate **primPaths** to identify the target prims. As **Layer** operates on the pseudo root, no prim paths are required.

Once a hierarchy level is selected, the settings are dynamically updated to offer the necessary parameters.

Metadata is defined using the **metadataName**, **metadataType**, and **value** fields. The **metadataType** dropdown is populated with all supported types in your USD version. For example, **String**, **Float**,or **Double**.

When setting metadata for attributes, use the **attributeNames** field to define a customizable list of attributes. Use the value in the first box to choose how many attributes you want to list.



A list of three attribute slots.

All types require the isCustom field to be set correctly to indicate if the metadata is pre-defined or a new, custom field.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
hierarchy	Prim	The USD entity type on which to set the metadata. Choose from **Attribute**, **Layer**, or **Prim**.
primPaths	N/A	Add the path(s) to the prims onto which to set the metadata. This property is not required when **hierarchy** is **Layer**
metadataName	N/A	The name of the metadata parameter.
metadataType	Token	The data type for the metadata parameter.
isCustom	No	Select **Yes** if this is a custom parameter and not one already offered by the schema.
value	N/A	The value to assign to the metadata.

UsdPayloadSet

The UsdPayloadSet node adds payloads to a scene. A payload is a special type of reference that can be optionally loaded or kept unloaded, providing a mechanism to defer the loading of certain, often resource intensive, parts of a scene until they are required. This allows for more efficient scene management and faster initial load times.

Unlike standard references that are loaded automatically when a scene is opened, payloads provide flexibility by letting users choose when to load them through the Payload working set. This is useful for large scenes where only a subset of the content might be needed immediately. Payloads which aren't loaded are still visible in renders (same principle as deferred loading).

To improve efficiency, Katana uses a deferred loading mechanism, for all data including USD. In the Katana Scene Graph tab, locations are only loaded and evaluated as you expand the graph. The same behavior has been introduced to the Scene Explorer tab and is used for both the USD Scene Graph and the Katana Scene Graph. However, if you need more control over this loading behavior with USD, you can use the Payload Working Set and Visibility Working Set. These give you full control over loading behaviour, or when locations are loaded.

Payloads are typically used for elements like background sets, props, or secondary characters that might not be needed immediately during scene setup or initial playbacks.

By using payloads, artists and TDs can more effectively manage and navigate complex scenes, ensuring resources are used efficiently and only loading heavy or less-critical data when it's genuinely needed.

Inputs

Connection Type	Connection Name	Function
Input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	N/A	Specify locations where payloads are applied. They are generated at each specified path, in the order in which the paths are listed.
asset	N/A	Loads in the attached file as a payload. This gets converted to the "asset/path.usda" section of a full payload path: @asset/path.usda@</prim/path>
assetPrimPath	N/A	The path to the prim that is to be used as a payload, located

Control (UI)	Default Value	Function
		inside the asset file. Or if no file is added, a prim path to a prim in the current stage can be used. This gets converted to the "prim/path" section of a full payload path: @asset/path.usda@</prim/path>
listPosition	prepend	Specifies how the payloads are added to the scene. Options are: • **append** - adds new payloads by placing them at the end of the list. This ensures that they are evaluated after existing values, giving them the potential to override preceding definitions. • **prepend** - adds new payloads to the front of the list, causing them to be evaluated before, and potentially overridden by values added later or in subsequent layers. • **delete** - removes the payloads from the list, preventing their properties from influencing the composed prim in the scene. • **reset to explicit** - clears any existing payloads on the prim and sets the payload list to be only the ones you're adding. It essentially instructs the USD scene to ignore any payloads on the prim(s) and only use the new ones. For more information, refer to the USD Glossary under List Editing.

UsdPrimCreate

In USD, a prim (short for primitive) is the fundamental building block of a scene. A prim represents a single element in the scene graph hierarchy and describes various types of entities and structures. Examples include points, cubes, materials, meshes, and xforms. The hierarchical locations of prims can be used to structure your scene before importing components into the prims.



Note: This node is the USD equivalent of Katana's PrimitiveCreate node.

UsdPrimCreate creates a prim defined by the **type**, which is selected from a list of available USD prim types. The **primSpec** specifier determines how the prim is treated during composition and has the following options:

- **define** -create the primitive in the scene.
- **over** - a prim that is used only to override opinions that exist in the scene.
- **class** - for defining default attributes using a class template.

By default, only the final, composed prim receives the new primitive using the **primSpec** specifier, with all others in the prim hierarchy affected using "over". This is to avoid overwriting the PrimSpecs in lower layers that go to compose the final prim. However, if you want to use the same specifier throughout the prim hierarchy, check **primSpecHierarchy**.

Katana interrogates the USD version in use to obtain a complete list of prim types to populate the **type** dropdown.

Once a type is selected, Katana then obtains all the properties for that prim type and populates the **properties** for further customization.



Note: For more information on prims, see Native USD Prims in the user guide, and Pixar's USD Glossary.



Note: There are also more specific nodes, such as UsdLight, UsdCamera, UsdScope and UsdXform nodes, which allow you to create different types of prims directly.

Inputs

Connection Type	Connection Name	Function
Input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	none	Specify locations where prims are to be created, such as /geo. Prims are generated at each specified path, in the order listed.
primSpec	define	Choose the specifier for the prim. This is the role or function of the prim in the scene description. • **define** - defines a new, complete specification for a prim and its properties within a layer. • **class** - defines a prim meant to act as a template for other prims. It ensures uniformity and reusability throughout the scene by allowing other prims to adopt or specialize it, inheriting its characteristics. • **over** - indicates that the prim is an override. It's used to non-destructively modify prims defined in lower-priority layers. A prim with an "over" specifier is meant to modify or extend the properties or hierarchy of existing prims, without redefining them entirely. For more information, refer to the USD Glossary under specifier.
type	none	Choose the type of prim to create at the position hierarchy defined by **primPath**.
primSpecHierarchy	off	If checked, create all prims in the **primPaths** hierarchy using the **primSpec**. If unchecked, all prims except the last one, will apply the **primSpec** in an "over" manner to avoid **primSpecs** in lower-order layers being overwritten.
properties	n/a	A dynamically generated list of properties and settings for the selected prim **type**.

UsdPrimvarSet

Primvars (short for primitive variables) are particular kinds of attributes that have two main functions:

- **Store data that varies through interpolation** - Primvars are used to hold data that varies over the surface of a primitive like UV coordinates or vertex colors.
- **Override attributes on a per-primitive basis** - Primvars can override attributes on a particular primitive, meaning you can customize the appearance or behavior of individual primitives within the scene hierarchy independently of others. This lets you change the details on a primitive that may be subject to shared attributes. For example, changing the color of one tree in a forest from green trees to red.

For example, consider a simple line defined by two points (vertices). You could define a color primvar to indicate that the color must change along the line from red to blue and that it is uniformly interpolated:

```
def BasisCurves "SimpleLine" {
    int[] curveVertexCounts = [2]  #Number of vertices
    point3f[] points = [(0, 0, 0), (1, 1, 1)] #Co-ordinates of the endpoints


    # The primvar for color:
    color3f[] primvars:color = [(1, 0, 0), (0, 0, 1)]  # Red to Blue
   uniform token primvars:color:interpolation = "vertex" #Interpolate the color
                                                                                #betwee
n the vertices
}
```

For more information see Primvar in Pixar's USD Glossary.

You can set a new primvar, or override the value of an existing one. Alternatively, you may need to remove a primvar's effect on the stage. These are achieved by selecting **Create/Override** or **Block** in the **action** dropdown.

Like Attributes, the value Primvars can be time-sampled, in that they can vary according to time. For this kind of primvar, set **multisample** to **Yes**.

Primvars offer a number of interpolation types to transition data between each data point and one is selected using **interpolation**. For more details on each, see the table below.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
action	Create/Override	Choose how you want to apply the primvar update. **Create/Override** - Create the primvar, or if it exists already, assign the new value. **Block** - Disable the primvar assignment in the prim(s), removing their effect from stage composition.
name	n/a	Assign a name to the primvar.
type	Double	Choose a data type for the primvar.
multisample	Yes	Select **Yes** if the data is time-sampled.
isCustom	No	Select **Yes** if this is a custom primvar.
interpolation	constant	Choose the interpolation type: • **Constant** - Uses only one constant value for the prim. For example, a single color applied to an entire mesh. • **Uniform** - Uses one value per geometric element, such as a face in a mesh. For example, if you have a cube and want each face to have a different color, you would use uniform interpolation.. • **Varying** - Uses one value for each vertex, but the value is smoothly interpolated over the whole face. This can an be used

Control (UI)	Default Value	Function
		for smooth color gradients across faces. • **Vertex** - Uses one value per vertex of the primitive. For a mesh, this could be colors or UVs specified at each vertex, leading to gradients across the faces. • **FaceVarying** - Uses one value per vertex per face. This is often used for UV coordinates and allows for each face of a mesh to have its own separate set of UVs or vertex attributes, even if the vertices are shared between faces.  **Note:** The names of these types are different to the convention used by Katana's arbitrary attributes' scope. For reference, see Arbitrary Attributes.
value	n/a	Assign a value to the primvar.

UsdPythonWrite

The UsdPythonWrite node lets you use Python in the **Parameters** tab to author USD layers. The resulting layer is composed with layers upstream.

 **Note:** The UsdPythonWrite node is not meant for standard workflow usage. We recommend you to use it for debugging and rapid testing.

Enabling the UsdPythonWrite Node

By default, the UsdPythonWrite node type is disabled. To enable it you must edit your launcher script.

For Windows

Copy and paste this environment variable into your launcher script.

```
set KATANA_ENABLE_USDPYTHONWRITE_NODE=1
```

For Linux

Copy and paste this environment variable into your launcher script.

```
export KATANA_ENABLE_USDPYTHONWRITE_NODE=1
```

Control (UI)	Default Value	Function
layerContent	layer.definePrim ("/root", "Scope")	A Python script to describe a USD scene.
Edit in External Editor	N/A	This feature allows you to edit your Python scripts using software other than Katana. This opens an external editor, as set up in **Edit** > **Preferences** > **externalTools** > **Editor**, for editing the script without blocking Katana's user interface. Under editor enter the location of the application you want to use. Katana monitors the text files you are editing, and when it detects that they have changed, it updates the node accordingly.

UsdReferenceSet

In Universal Scene Description (USD), references are fundamental composition operations used to construct scenes using a modular approach. They enable the reuse and assembly of content by incorporating the contents of one layer into another layer. This reuse goes beyond simple import and offers powerful overrides and variations when combined with other composition operations.

Artists can add external USD files, like assets or scene fragments, to a scene using references. This promotes reusability and flexibility, enabling the use of a single asset across multiple scenes without making destructive edits.

Example: When an artist makes a generic tree asset as a USD file, they can reference this tree in different environments or scenes. If they want to create a forest, they can reference the same tree multiple times with

different overrides. This creates variation in size, color, or form, resulting in a diverse forest using just one base asset.

References facilitate layered composition and versioning by allowing localized changes without modifying the original asset file. In essence, they help efficiently manage and deploy assets in complex scenes, ensuring consistent updates and enabling targeted modifications.

To help understand how references are defined, consider referencing a simple sphere primitive from a scene file. Imagine the sphere is called blueSphere and has been defined in a file called **aSphere.usda**. Our simple scene file could reference the file as follows:

```
#usda 1.0
def "SceneRoot" {
 references = @./aSphere.usda@blueSphere
}
```

Setting USD References

Use UsdReferenceSet to add references to your scene that refer to other prims..

Prims to contain the references are added to **primPaths**.

You can then browse to a USD asset file using the **asset** field. Once selected, the **assetPrimPath** is populated with the scene hierarchy for the asset, allowing you to select a prim to reference.

Once the reference has been selected, you can decide how to place the reference in relation to existing references on the prim using the **listPosition** dropdown. Choose to **append** to the end of the existing list of references, **prepend** to the start, or clear the existing references and replace with the ones you're adding **(reset to explicit)**. Alternatively, choose **delete** to remove the references that you've specified in the node parameters.

Controls

Control (UI)	Default Value	Function
primPaths	N/A	Specify locations where references are applied. References are generated at each path, following the order in which they are listed.
asset	N/A	Load a USD file to use for references. This gets converted to the "asset/path.usda" section of a full reference path:

Control (UI)	Default Value	Function
		@asset/path.usda@</prim/path>
assetPrimPath	N/A	The path to the Prim to be used as a reference. All prims in the **asset** file are given in their scene hierarchy. If no reference file is added, you can use the prim path to a prim in the current stage. This becomes the "prim/path" part of a full reference path, in this format: @asset/path.usda@</prim/path>.
listPosition	prepend	Specifies how the references are added to the scene. Options are: • **append** - adds the new reference(s) by positioning them at the end of the current list, ensuring they are evaluated after existing values, allowing them to potentially override preceding definitions. • **prepend** - adds the reference(s) to the front of the list, causing them to be evaluated before, and potentially overridden by values added later or in subsequent layers. • **delete** - removes one or more references from the list, preventing their properties from influencing the composed prim in the scene. **reset to explicit** - clears any existing references on the prim and sets the reference list to be only the new reference(s) you're adding. It essentially instructs the USD scene to ignore any previous references on the prim(s) and only use the new ones For more information, refer to the USD Glossary under List Editing.

UsdRelationshipSet

UsdRelationshipSet sets a relationship on a prim. In USD, a relationship (or relationship property) is a directed connection between a prim and a property. It helps to establish references, dependencies, and links

within the scene graph, avoiding data duplication. Instead of duplicating data in multiple locations, a reference points to the source. Any updates made to the source are then reflected in the referring prim.

Example: For example, let's say you have a skyscraper called "MegaStructure" with multiple interchangeable lobby designs. Instead of storing lobby details within the "MegaStructure" prim, we can use a relationship to reference a particular lobby layout. For example, **rel lobbyDesign = </path/to/futuristicLobbyAsset>**.

This relationship has the name lobbyDesign and its value is the path to the choice of lobby asset. This allows "MegaStructure" to dynamically associate with the futuristic lobby design. If the design preference changes, the relationship can easily point to a different lobby asset, enabling simple swaps and modifications while optimizing data storage and scene management.

Custom and Pre-Defined Relationships

In a USD schema, relationship properties may already exist to construct the prim in accordance with the prim schema. Alternatively, a custom relationship property is not predefined by any schema. Instead, it is added to meet specific needs. They allow the expansion of properties of a prim beyond what the predefined schemas offer.

When adding relationship, you can tell Katana if it is a custom relationship by setting **isCustom** to **Yes**. You should choose the correct option to ensure correct operation.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	N/A	The list of prim paths to receive the relationship. Manage this list using the **Add Locations** menu.
relationshipName	N/A	Assign a name to the relationship.
isCustom	No	Select **Yes** if this is a custom relationship.
value	N/A	The destination of the relationship. It can be a path to another prim, multiple prims, or specific properties. The target path is represented in a string format as </path/to/targetPrim>.Manage this list using the **Add Locations** menu.

UsdSchemaSet

A USD schema is structured data that determines the role of a prim on the stage. Schemas provide a framework to define common types and are essentially the blueprint for specific objects or behaviors within the stage.

The schema specifies attributes such as size, height, display color, and other characteristics that make up the object.

Creating a prim of a certain type directly assigns a predefined schema, dictating its properties and behavior.

The UsdSchemaSet node associates an additional type or category of behavior and data with a prim. By setting the schema in the node, you indicate that this specific prim should be handled using an additional defined set of attributes and behaviors particular to that schema.

For example, you could use UsdSchemaSet to add shadow properties to a light giving it attributes that it normally wouldn't have.

With the UsdSchemaSet you are setting a schema using an API (like a MaterialBindingAPI or VisibilityAPI). API schemas ending in "API" are designed to be applied to prims to add supplementary capabilities or attributes to an existing prim non-destructively (without altering the original type of the prim). This flexibility allows artists and TD's to create more complex and layered scene descriptions while maintaining a clear and efficient structure.

There are many schemas available such as lights, shadows, render-specific attributes, and you also have the flexibility to create custom schemas to suit your specific needs.

Example: Imagine you have a light in your 3D scene. By default, this light doesn't cast shadows. If you want it to, you can use the UsdSchemaSet to add a "ShadowAPI" schema, providing new attributes related to shadows. This way, without changing the original light, you add the ability for it to cast shadows in your scene, showcasing how schemas in USD let you flexibly add features to prims.

In the UsdSchemaSet node, you can pick an API Schema from a list that automatically updates itself. This keeps the list current, even if new items are added after setting up Katana or with updated USD versions. This way, you always see the most up-to-date options available and can integrate schemas beyond Katana's original set.

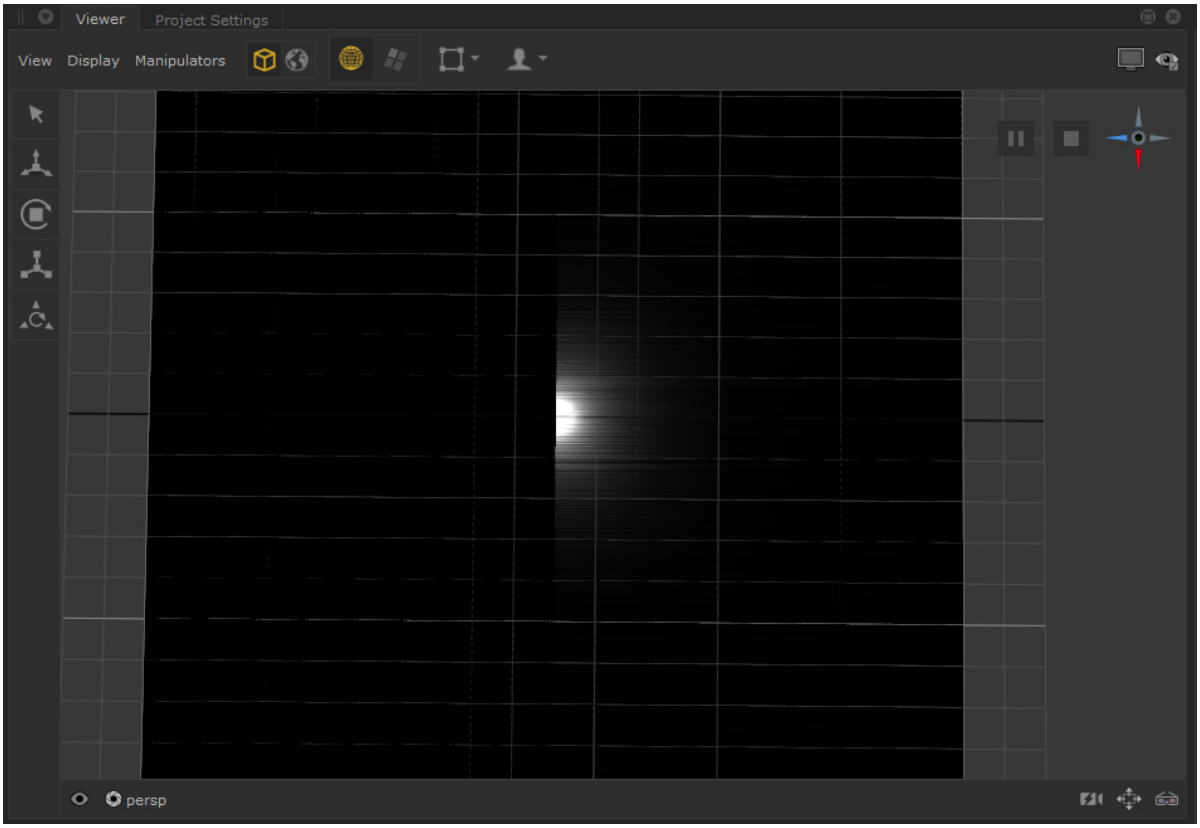


Note: For more information on USD schema see The USD Glossary.

Apply Schemas to a Prim

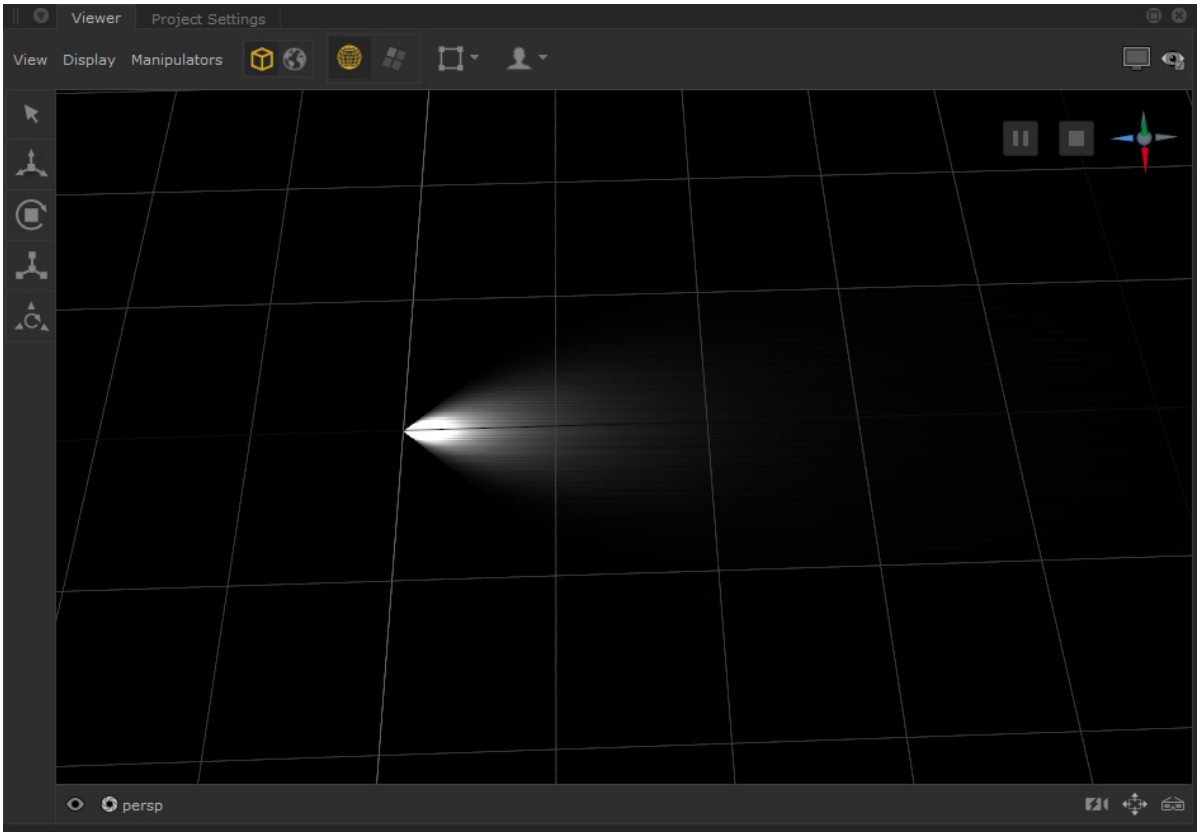
You can apply multiple schemas to a prim. For example, if we take a diskLight created with UsdPrimCreate we can add a cone shaping by applying a schema.

1. In the **viewer** monitor select **display** and untick **default lighting**.
2. Create a plane using another **UsdPrimCreate**, specify the **primPath** (for example /bg) but this time set the **type** to **Plane**. This will be the background for the light to shine on. Without it the light would be invisible.
3. Set the plane's **axis** to **Y** under **properties** and increase **length** and **width** to **10**.
4. Let's now create a **UsdPrimCreate** node and in **primPaths** specify the path where your new diskLight prim will reside in the USD scene hierarchy. (For example /light)
5. Set the **type** parameter to **DiskLight**.
6. Increase the **exposure** property of the diskLight under **properties**>**inputs**. For example, set it to **20** to enhance it's visibility.



You can then apply a ShapingAPI on top of the light using the UsdSchemaSet node. To do this follow these next steps:

1. Create a **UsdSchemaSet** node, then drag and drop the diskLight prim path (in this example /light) from the **Scene Explorer** into **primPaths** using your middle mouse button.
2. Change **type** to **ShapingAPI**.
3. Then adjust the **angle** in **properties**>**inputs**>**shaping**>**cone** and the **focus** in >**shaping** to control the cone's opening and achieve your desired light shape.



This now means that the diskLight, which already has properties like diffuse, color, intensity, exposure will also inherit all the attributes of the ShapingAPI, such as angle, softness, focus, and other ShapingAPI properties. The diskLight now has an enhanced set of properties, combining both its inherent attributes and those inherited from the ShapingAPI, enabling more nuanced control over its visual appearance and behavior in the scene.

Controls

Control (UI)	Default Value	Function
primPaths	none	Specify locations where API Schemas are applied. Schemas are generated at each specified path, following the order in which they are listed.
type	none	Choose the type of API Schema to create from the list. This list is updated in real-time. This ensures it includes newer schemas beyond Katana's original set, providing you with the latest options.
listPosition	prepend	Specifies how the API Schema is added to the scene. Options

Control (UI)	Default Value	Function
		are: • **append** - integrates new schemas by positioning them at the end of the list, ensuring they are evaluated after existing values, and thereby, allowing them to potentially override preceding definitions. • **prepend** - adds one or more schemas to the front of the list, causing them to be evaluated before, and potentially overridden by values added later or in subsequent layers. • **delete** - removes one or more schemas from the list, preventing their properties from influencing the composed prim in the scene. • **reset to explicit** - Ignoring all previously compiled schemas, this operation sets the list strictly to the defined value(s), providing a clear and explicit new starting point for schema evaluation. "list" specifically refers to the finalized list of all values and properties applicable to a prim, derived after evaluating all USD layers and compositional arcs. For more information, refer to the USD Glossary under List Editing.
properties	n/a	A list of dynamically generated properties and settings depending on which **type** of API Schema type is selected.

UsdScope

The UsdScope node enables you to create scope prims. This can act as a wrapper for Scope from the UsdPrimCreate node, helping organise your node graph and improve workflow for setting up locations in USD.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	N/A	Location Paths of where to create Prims. For example, /root/world/geo. Prims will be created at each of the provided paths in the order they are provided.
primSpec	define	Determines what specifier of Prim to create, from **define**, **class** or **over**.
primSpecHierarchy	disabled	Determines whether to create all prims in the provided hierarchy with the primSpec. If set to False, all but the final Prim will be "over" primSpec to avoid writing over primSpecs in lower order layers.

UsdSpecializeSet

Specializes are a composition arc that allows one prim to "specialize" another prim. This means the specializing prim inherits the structure and properties of the prim it specializes, but it doesn't replace or override it. The specializes relationship is a means to establish a "class-like" inheritance mechanism within USD.

Like inheritance, you can create a specialized prim from another prim. However, the overrides on the specialized prim level will always take precedence over any further references. This can be useful to ensure certain aspects are never overridden.

Unlike references, which can replace a prim with content from another USD file, specializes allows one prim to inherit the attributes and children of another without replacing it.

Multiple prims can specialize a single prim, allowing layered inheritance where each specializing prim can further refine or extend the properties of the original.

Specializes are often used to set up a base set of properties or structure that can be shared across multiple prims. Think of it as setting up a "class" in programming, where multiple objects can inherit properties and methods from the class but can also have their unique attributes.

It's a powerful tool in the composition toolkit of USD, allowing for the setup of shared structures and properties that can be extended in a modular and non-destructive manner.

Example: Consider constructing digital models of cars using USD. There's a fundamental "car" prim, which describes generic attributes like having wheels, and seats.

Now, we wish to create specialized versions of this car, such as a convertible and truck. Employing the "specializes" relationship in USD, both the convertible and the truck inherit the foundational attributes from the car prim, ensuring they have wheels, and seats. Yet, they bring in their distinct characteristics like a retractable roof for the convertible and extra space for the truck.

If a modification is made to the basic car prim, such as changing the wheel design, both the convertible and truck automatically incorporate the change while preserving their individual specialties. This approach ensures a consistent base among all car models while allowing each specialized version to express its unique features.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	N/A	Specify locations where specializes are applied. They are generated at each specified path, following the order in which they are listed.
specializePath	N/A	The path to a Prim you wish to specialize from.
listPosition	prepend	Specifies how the specializes are added to the scene. Options are: • **append** - adds the new specializes by putting them at the end of the current list, ensuring they are evaluated after existing values,. This allows them to potentially override preceding definitions. • **prepend** - adds the new specializes to the front of the list, causing them to be evaluated before, and potentially overridden by, values added later or in subsequent layers. • **delete** - removes one or more specializes from the list, preventing their properties from influencing the composed prim in the scene. • **reset to explicit** - ignore all previously compiled specializes and set the to only the new value(s), providing a clear and explicit new starting point for specializes evaluation. For more information, refer to the USD Glossary under List Editing.

UsdStageDefine

The UsdStageDefine node enables you to set up stage-specific parameters before starting to build your stage in USD.

If you are using a custom ArResolver in their production, this allows you to provide additional data to the resolver for use during asset path resolution.

Inputs

Connection Type	Connection Name	Function
This node has to be set at the top of the node graph, hence this node has no inputs.		

Controls

Control (UI)	Default Value	Function
populationMask	N/A	Sets the population mask for the stage. The Prim paths will make up a population mask of the prims populated in the stage. If this is empty, we do not apply, and clear, any population mask on the stage.
assetResolverContexts	N/A	The Asset Resolver 2.0 contexts provided to a stage in order to process Asset Resolution within the context set with this stage. Stored as a list of pairs. The first element in the pair is the URI/IRI scheme for the ArResolver that will be used to create the ArResolverContext from the second element in the pair. An empty resource identifier scheme will utilize primary resolver.

UsdSubLayerAdd

A layer is a foundational element in the USD stage and is typically a file containing scene information. The file can be a .usda (ASCII) or .usdc (binary) file, though the more generic .usd extension may be used for either type.

Sublayers allow artists to build complex scenes by layering various USD files together, each one potentially containing different elements or variations of a scene. Sublayers are ordered in a stack, and they are composed together based on this order, enabling artists to manage, override, and organize scene content in a non-destructive and scalable manner.

This structuring allows for efficient collaboration across departments, as different aspects of a scene (like lighting, animation, or shading) can be worked on in parallel and then combined together cohesively.

Layers can be further broken down into sublayers, where content is organized into its parts, each stored in a separate USD file.

Example: The layers could be separated as follows:

Layout layer - Items are brought into their initial positions on the stage.
Animation layer - Allows your animation department to override any characters or props with animation data.
FX layer - FX is added to bring additional motion and life to the scene.
Lighting layer - Lights are added and any final rendering optimizations are applied.

Each of these layers can have both an entire sequence and a per-shot level version.

Layer Interaction

Layers are organized in a stack, where the topmost layer can override the data in the layers below. This enables non-destructive editing, as the data can be overridden but not lost.

Given the fact that data can be overridden, it is not absolutely certain that it will be used. Consequently, the data is known as an "opinion". As part of scene composition, opinions are resolved to establish the data to use depending on the strength of the opinion. This strength is determined by a prescribed set of rules including layer order.

The various mechanisms that are used to combine layers and resolve data are known as the "composition arc", which essentially determines how everything fits together to produce the final composition.

Organizing Layers

The UsdSubLayerAdd node resembles the functionality of the UsdIn node but expands its capabilities with layer separation. You can create separate sublayers like "fx," "layout," and "lighting" to organize your project into department-specific layers.



You can also divide these sublayers into additional sublayers to accommodate different versions, variants, or sequence-based effects, forming multiple branches from this structure.

Plus, sublayers offer you the flexibility to selectively choose which data to override at different points in your project.

Layering sublayers into layer stacks is especially helpful for loading root-level layers without requiring a specific prim path. This feature is useful for incorporating entire stages, or layering prims that share the same hierarchy.



Note: For more information, refer to the USD Glossary under SubLayers.

Inputs

Connection Type	Connection Name	Function
Input	in	The incoming scene graph data that the node will operate on or modify.

Control (UI)	Default Value	Function
asset		Loads in a file as a sublayer. • **Browse...** - open a file browser to locate the USD file. • **Set Node Name from Path** - automatically rename your node to match the name of the imported asset.

UsdToKatana

This node lets you convert upstream USD data into Katana data for rendering or editing the scene using Katana's range of Geolib nodes.



Note: The KatanaToUsd node converts Geolib data to USD data without the need to bake out layers, enabling you to handle multiple data types. Complete the USD "round-trip" alongside UsdToKatana node, so you can leverage Katana's pre-existing artist tools and convert back to USD data afterwards if you so wish.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
deactivatePrims	enabled	Specify the prim path(s) below which all USD data is loaded (including the specified paths).
isolatePath	n/a	Load only the USD contents below this prim path.
motionSampleTimes	n/a	Specifies the motion sample times to load. The default behavior is no motion samples, and load only the current time 0.
instanceMode	expanded	When set to **expanded**, instances are loaded as though the children of parent primitives were there naturally. When set to **sources and instances**, parents of instanced primitives are created under a sibling of **/world** named **/instances**.
usePurposeBasedMaterialBinding	No	When enabled, it allows you to specify material bindings to be loaded in for objects with a purpose tag from the USD stage. When disabled, Katana will load in the whole USD scene with all material bindings resolved.

Control (UI)	Default Value	Function
		Note: For more information, see USD Glossary: Purpose.
prePopulate	enabled	Enable to load all payloads and pre-populate the stage. (Assuming the entire stage will be needed). Enabling this is more efficient since USD can use its internal multithreading.
verbose	disabled	When enabled, the information generated during USD scenegraph generation is output to the command line.
evaluateUsdSkelBindings	enabled	Enable to apply skinning to each prim that is bound to a skeleton.

UsdTransformEdit

The UsdTransformEdit node enables you to edit certain transformOps. By filling in the **xformOpName**, this will populate the translate, rotate and scale with its values. When values are changed, this will overwrite the transformOp.

To learn more about transformations, see Native USD Transformations.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPath	N/A	Specify the prim which you want to transform.
xformOpName	N/A	Specify the TransformOp to be edited.
order	Scale Rot Trans	Select the order by which the three transformations are applied.
rotateOrder	Rx Ry Rz	Select the order by which the three transformations are applied.
translate	0.0	The amounts to translate along the x, y, and z axes.
rotate	0.0	The amounts to rotate the prims around the x, y, and z axes.
scale	1.0	The amounts to scale the prims along the x, y, and z axes.
pivotTranslate	0.0	The amounts by which the pivot point for transformations is translated along the x, y, and z axes.
pivotRotate	0.0	The amounts by which the pivot point for transformations is rotated along the x, y, and z axes.
makeInteractive	Yes	When set to "Yes", the selected path can be interactively manipulated in the Viewer. This option is only available when the mode parameter is set to "Single Path".

UsdTransformSet

In USD, a "transform" commonly refers to the spatial transformation of a prim in the scene. Transformations are used to position, orient, and scale prims in 3D space. The main types of transformations are translation, rotation, and scale. You can also set a pivot point from which translation or rotation occurs. To learn more about transformations, see Native USD Transformations.

UsdTransformSet applies a 3D transform to one or more prims as defined by their paths in the USD scene. You can choose to append the transform to existing transforms using **appendTransform**, or leave it unchecked to overwrite any transforms on that prim.

Enter transform values along the x, y, and z axes for translate, rotate, and scale. You can also enter values for the pivot point for transformations away from the origin by translation and rotation.

The three transformation types can be applied in order as defined by any permutation of scale, rotate, and translate. Select this using the **order** dropdown.



Note: UsdTransformSet is the USD equivalent of Transform3D. However, there are differences between the default settings. For example, the stack order in Geolib defaults to "first", whereas the defaults in USD are the opposite.

The UsdTransformSet node also has an option which allows you to use Viewer transform manipulators against USD native scene locations. To use this option, **makeInteractive** should be set to **Yes**. See Viewer Manipulators for Native USD to learn more about the workflow.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPaths	N/A	The prim locations as described by their paths in the USD scene hierarchy
xformOpName	same as node name	The name given to the node. This is assigned automatically as the node name to make it unique. You can override the default to allow overrides for specific transform Ops.

Control (UI)	Default Value	Function
		The node name is set using the 'nodeName' expression. This can be edited if you prefer.
appendTransform	off	Check this to append the transform to the current list of transforms that are active on the prim(s). If unchecked, the transform overwrites any existing transforms on the prim(s).
order	Scale Rot Trans	Select the order by which the three transformations are applied.
rotateOrder	Rx Ry Rz	Select the axes order by which the rotation takes place. By default the rotation is first around the x axis, then y, and finally z.
translate	0.0,0.0,0.0	The amounts to translate along the x, y, and z axes.
rotate	0.0,0.0,0.0	The amounts to rotate the prims around the x, y, and z axes.
scale	1.0,1.0,1.0	The amounts to scale the prims along the x, y, and z axes.
pivotTranslate	0.0,0.0,0.0	The amounts by which the pivot point for transformations is translated along the x, y, and z axes.
pivotRotate	0.0,0.0,0.0	Enter the amounts by which the pivot point for transformations is rotated around the x, y, and z axes.
makeInteractive	Yes	When set to "Yes", the selected path can be interactively manipulated in the Viewer. This option is only available when the mode parameter is set to "Single Path".
asMatrix	disabled	If makeInteractive is set to "No", this option becomes visible, allowing you to set the attributes as a matrix.

UsdVariantSet

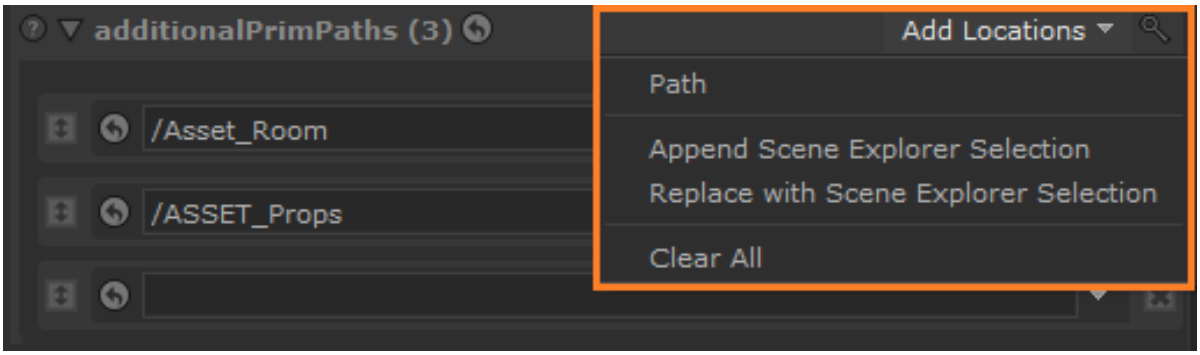
USD's robust support for non-destructive editing offers various features for managing versions of prims. This includes the ability to version parts of the scene while preserving the original data using variants. A variant allows multiple alternative versions of scene data to coexist in the same stage. They enable artists and

technical directors to handle multiple versions of a prim or a group of prims without having to duplicate the entire asset.

Key to working with variants is the concept of a VariantSet. This is a convenient container that holds groups of variants for a prim or prims. For example, you might place different lighting variants in a VariantSet called "DuskLighting", holding several options for different lighting styles that you could use in a dusk-lit scene.

UsdVariantSet lets you select a variant for a prim from a VariantSet. The variantSet dropdown is automatically populated with any variant sets associated with the prim location in primPath. Select from variantSet to populate the variant dropdown with all the variants contained in the set. You can then choose a variant to apply to the prim from the variant dropdown.

Note: The node UsdVariantSet node is named consistently with the other "set" nodes and refers to setting the variant on the prim, not the noun for a collection, "set".



Use Add Locations to manage the list of additional prims for the variant.

- **Path** - Add an entry path field to populate as required.
- **Append Scene Explorer Selection** - Add a new path field to the list containing the current selection in the Scene Explorer.
- **Replace with Scene Explorer Selection** - Replace all current path entries with the selection from the Scene Explorer.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
primPath	n/a	Specify the location of a prim that is to be assigned the variant.
variantSet	n/a	Choose a VariantSet from the ones available on the prims in **primPath**.
variant	n/a	Choose a variant from the list of those in the set named in **variantSet**.
additionalPrimPaths	n/a	Add any additional prims that should use the variant. Use the **Add Locations** menu (see above) to manage this list.

UsdXform

The UsdXform node enables you to create Xform prims. You can then transform using this node.

Inputs

Connection Type	Connection Name	Function
input	in	The incoming scene graph data that the node will operate on or modify.

Controls

Control (UI)	Default Value	Function
mode	Single Path	The prims to be transformed may be specified either using a single path or multiple. Note: The "makeInteractive" parameter is currently only available when this option is set to "Single Path".
primPath	N/A	The prim path where the prim will be located.
primSpec	define	Determines what specifier of Prim to create, from **define**, **class** or **over**.
xformOpName	UsdXform	The name to be given to the XformOp when stored in USD.
order	Scale Rot Trans	Select the order by which the three transformations are applied.
rotateOrder	Rx Ry Rz	Select the axes order by which the rotation takes place. By default the rotation is first around the x axis, then y, and finally z.
translate	0.0	The amounts to translate along the x, y, and z axes.
rotate	0.0	The amounts to rotate the prims around the x, y, and z axes.
scale	1.0	The amounts to scale the prims along the x, y, and z axes.
pivotTranslate	0.0	The amounts by which the pivot point for transformations is translated along the x, y, and z axes.
makeInteractive	Yes	When set to "Yes", the selected path can be interactively manipulated in the Viewer. This option is only available when the mode parameter is set to "Single Path".

Miscellaneous Nodes

The nodes in this section are considered Miscellaneous and fall under the **Misc** category in Katana. These are listed alphabetically, and each node includes a short description followed by a list of the node's parameters and their functions.

SuperTool Nodes

The following section describes Katana's 3D **SuperTool** nodes.

GafferThree

The GafferThree node type allows you to create lights under an arbitrary hierarchy of rigs. Materials, transformations, and constraints can be applied to lights from within the GafferThree's **Parameters** interface. This node also supports:

- Creating and applying Template Materials to lights.
- Muting and soloing lights, or all lights under a rig.
- Linking lights to specific objects.
- Editing lights from the incoming scene, and editing multiple lights at once.
- Adding aim constraints to lights.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to create a GafferThree for creating several lights together.

Control (UI)	Function
 > Select In Scenegraph	Sets the scene graph path to the location to be created. For more information, see Common Parameter Widgets.
 > Show Incoming Scene	Displays all incoming lights, rigs, and Template Materials from upstream Gaffer-type nodes.
sync selection	When enabled, selecting a Gaffer light within the **Parameters** tab selects its location within the **Scene Graph** tab: • **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.
[Gaffer object table]	Displays a list of all objects controlled by this GafferThree node. The object table contains the following information: • **Name** - the name of the object. Double-click in this column to change the name of the item. • **M** - click to mute the object so that it is omitted from renders. • **S** - click to solo the object so that everything not solo-ed is omitted from interactive renders. • **Shader** - displays the shader associated with the object. You can also right-click in this column to select a shader. Once you've added a shader, double-click in this column to assign or change it at any time. • **Color** - specifies the color of a light. Double-click the swatch to activate the color picker. If there isn't a swatch, you need to add a color in the **Material** tab before you can change it in this column. • **Int** - sets the light intensity. • **Exp** - sets the light exposure. • **Linking** - indicates whether or not the item is linked. A star in the entry indicates there are exceptions.
Right-click the [Gaffer object table item]	
Add >  Template Material	Adds a Template Material to the gaffer table. The Template Material is assigned to lights, providing the same material for multiple lights. Each light is

Control (UI)	Function
	also capable of overriding the defaults set by this Template Material.
Add >  Light	Adds a light to the object table.
Add >  Rig	Adds a rig for multiple lights to the object table.
Add >  Light Filter Reference	Adds a light filter that references another light filter package.
Add >  Light Filter	Adds a light filter that modifies the behavior of a light, depending on the renderer you're using.
Add >  Sky Dome	Adds a sky dome light, used to provide environment lighting to your scene.
Add > Import Rig...	Adds a previously exported rig to the object table.
Delete	Deletes the selected entity in the object table.
Duplicate	Creates a copy of the currently selected entity.
Adopt for Editing	Allow you to make edits on a light, rig, or Template Material that has been shown from an incoming scene, which can be any upstream Gaffer-type node.
Delete Edit Package	If you adopted a light, rig, or Template Material for editing, you can revert back to a read-only state and reverse the changes that you applied.
Toggle Lock State of Selected Items	Toggles the lock state of the selected entity in the object table.
Group Selected Siblings Under Rig	Groups the selected siblings under a newly created rig.
Export Rig	Exports the currently selected item as a **.rig** file.
Create Shared Light Filter	Creates a light filter on a light that references a light filter that already exists in another location in the scenegraph.
Expand All	Expands the selected branch in the object table to reveal all children. If the selected branch does not have any children, nothing happens when attempting to expand.

Control (UI)	Function
	Note: In the menu, **Expand All** becomes **Expand Branch** whenever there is more than one item in the Gaffer object table.
Expand All To	Expand the branch to a specific type, either assembly, component, or level-of-detail group. This method of expansion applies specifically to the scene graph, and has limited use for the GafferThree. **Note:** In the menu, **Expand All To** becomes **Expand Branch To** whenever there is more than one item in the Gaffer object table.
Collapse All To	Collapse the branch to a specific type, either assembly, component, or level-of-detail group. This method of collapse applies specifically to the scene graph, and has limited use for the GafferThree. **Note:** In the menu, **Collapse All To** becomes **Collapse Branch To** whenever there is more than one item in the Gaffer object table.
Collapse All	Collapses the selected branch in the object table to hide all children. If the selected branch does not have any children, nothing happens when attempting to collapse. **Note:** In the menu, **Collapse All** becomes **Collapse Branch** whenever there is more than one item in the Gaffer object table.
Expand Location	Expands the selected location to only the children and leaves directly below it in the hierarchy.
Collapse Location	Collapses the selected location and any children and leaves directly below it, but not any entities higher than the location in the hierarchy.

The display parameters for the **Object**, **Material**, and **Linking** tabs are dependent on what's selected in the Gaffer object table. Where a particular tab isn't listed for an object type, there are no parameters in that tab.

Control (UI)	Default Value	Function
Object list: Template Material		
Material tab		
useLookFileMaterial	Disabled	When enabled, the material from an associated Look File is used.
Add Shader	N/A	Click to add a renderer-specific shader from the dropdown list. The **Material** tab is populated with controls appropriate to the shader selected, and are dependent on the renderers installed.

Control (UI)	Default Value	Function
Object list: light		
Object tab > geometry		
projection	perspective	Sets the light projection mode: • **perspective** - a warped projection where distant objects/features appear smaller than those nearer the camera. • **orthographic** - a two-dimensional representation of a three-dimensional object.
radius	1	Sets the light's radius.
fov	70	Controls the field of view angle in degrees.
orthographicWidth	30	Sets the orthographic projection width.
centerOfInterest	20	Sets the center of interest.
near	0.1	Sets the near clipping plane distance.
far	100000	Sets the far clipping plane distance.
screenWindow	-1.0, 1.0, -1.0, 1.0	Controls the screen window placement on the imaging plane. They are the **left**, **right**, **bottom**, and **top** bounds of the screen window.

Control (UI)	Default Value	Function
Object tab > transform		
transform	N/A	Transforms the light according to the SRT or matrix controls. For more information, refer to the Transform Controls Widget Type in the Common Parameter Widgets.
transform > Tools	N/A	Adjusts the light to match selected scene graph selection options in the dropdown menu. For more information, refer to the Transform Tools Widget Type in the Common Parameter Widgets.
enable aim constraint: enabled; aim constraint options		
targetPath	N/A	Specifies the object(s) to constrain to. If you want to aim a light to point at a target, this is the target. If you set multiple targets, then the constraint aims at the average center of the objects. The **targetPath** parameter options are available in either a scene graph widget or dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
addToConstraintList	Yes	Specifies whether or not to add the base path for the light to the globals.constraintList at **/root/world** in the **Attributes** tab.
targetOrigin	Object	Sets how the center of the target object is calculated: • **Object** - uses the local origin of the object as the target. • **Bounding Box** - uses the center of the object's bounding box as the target. • **Face Center Average** - uses the face center average of the object as the target. • **Face Bounding Box** - uses the face center average of the object's bounding box as the target.

Control (UI)	Default Value	Function
baseAimAxis	0.0, 0.0, -1.0	The axis of the base object that is pointed at the target. Adjusting these values changes the side of the object that is aimed at the target.
baseUpAxis	0.0, 1.0, 0.0	The axis of the base object that is pointed upwards relative to the target. Adjusting these values changes the rotation of the base object, while keeping the aim constant.
targetUpAxis	0.0, 1.0, 0.0	The axis of the target object that is pointed upwards relative to the base object. Adjusting these values changes the rotation of the target object, while maintaining the aim constant.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.
Object tab > annotation		
text	N/A	Places a label in the Viewer containing the string entered in the text field.
previewColor	1.0, 1.0, 1.0	Specifies the color of the light annotation in the Viewer. This value does not affect the color value of the light. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
Material tab > material		
useLookFileMaterial	Disabled	When enabled, the material from an associated Look File is used.
Add Shader	N/A	Click to add a renderer-specific shader from the dropdown list. The **Material** tab is populated with controls appropriate to the shader selected, and are dependent on the renderers installed.
useLookFileMaterial: enabled		

Control (UI)	Default Value	Function
reference > asset	N/A	Set the path to the asset you want in your scene. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
reference > materialPath	N/A	Choose the material path from the dropdown list, based on the asset you have listed in the **asset** field above. If you do not have an **asset** listed in the asset field, nothing appears in the dropdown list for **materialPath**.
Linking tab > light linking Support for light linking is dependent on your renderer.		
action	append linking information	Determines whether the linking options set in this node override the incoming scene options or if the new settings are appended to the incoming options. If this light doesn't exist in the incoming scene, this option has no effect.
initialState	on (or use existing value for adopted lights)	Determines whether the newly-added light location is initially **on**, **off**, or **use existing value**. **Note:** The **use existing value** option is only available for adopted lights.
clearUnmatched	disabled	When linking is resolved, the **clearUnmatched** parameter determines whether or not existing light linking attributes for this light are removed from locations that do not match the **on** or **off** expressions. The effect of this parameter is only visible in the **Attributes** tab when linking has been resolved, which means after a LightLinkResolve node or when Implicit Resolvers are active. Examines the **lightList** attribute on your linked objects to ensure that the attributes have been set correctly. If the parameter has been disabled, the value of the **enable** child

Control (UI)	Default Value	Function
		attribute in the **lightList** attribute for your light is **0**; otherwise, the default enabled setting is **1**.
on	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning the light **on** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to CEL Statement Widget Type in Common Parameter Widgets.
off	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning the light **off** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using the CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to CEL Statement Widget Type in Common Parameter Widgets.
Linking tab > shadow linking Support for shadow linking is dependent on your renderer.		
action	append linking information	Determines whether the linking options set in this node override the incoming scene options or if the new settings are appended to the incoming options. If this light doesn't

Control (UI)	Default Value	Function
		exist in the incoming scene, this option has no effect.
initialState	don't set value (or use existing value for adopted lights)	Determines the initial value for shadow visibility in the **lightList** entry for the newly-added light: **on**, **off**, **don't set value**, or **use existing value**.  **Note:** The **use existing value** option is only available for adopted lights.
clearUnmatched	disabled	When linking is resolved, the **clearUnmatched** parameter determines whether or not existing light linking attributes for this light are removed from locations that do not match the **on** or **off** expressions. The effect of this parameter is only visible in the **Attributes** tab when linking has been resolved, which means after a LightLinkResolve node or when Implicit Resolvers are active. Examines the **lightList** attribute on your linked objects to ensure that the attributes have been set correctly. If the parameter has been disabled, the value of the **enable** child attribute in the **lightList** attribute for your light is **0**; otherwise, the default enabled setting is **1**.
on	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning shadow casting from this light **on** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to the CEL Statement Widget Type in Common

Control (UI)	Default Value	Function
		Parameter Widgets.
off	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning shadow casting from this light **off** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to the CEL Statement Widget Type in Common Parameter Widgets.

Control (UI)	Default Value	Function
Object list: rig		
Object tab > transform		
transform	N/A	Transforms the rig according to the SRT or matrix controls. For more information, refer to the Transform Controls Widget Type in the Common Parameter Widgets.
transform > Tools	N/A	Adjusts the rig to match selected scene graph selection options in the dropdown menu. For more information, refer to the Transform Tools Widget Type in the Common Parameter Widgets.
Object tab parameters continued		
enable point constraint	disabled	When enabled, specifies the **point constraint options**.
enable orient	disabled	When enabled, specifies the **orient constraint options**.

Control (UI)	Default Value	Function
constraint		
when enable point constraint: enabled; point constraint options		
targetPath	N/A	Specifies the object(s) to constrain to. If you want to aim a rig to point at a target, this is the target. If you set multiple targets, then the constraint aims at the average center of the objects. The **targetPath** parameter options are available in either a scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
addToConstraintList	Yes	Specifies whether or not to add the base path for the rig to the globals.constraintList at **/root/world** in the **Attributes** tab.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
baseOrigin	Object	Sets how the center of the base object is calculated: • **Object** - uses the local origin of the object as the origin. • **BoundingBox** - uses the center of the object's bounding box as the base origin.
targetOrigin	Object	Sets how the center of the target object is calculated: • **Object** - uses the local origin of the object as the target origin. • **BoundingBox** - uses the center of the object's bounding box as the target origin. • **FaceCenterAverage** - uses the face center average of the object as the target origin. • **FaceBoundingBox** - uses the face center average of the object's bounding box as the target origin.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path.

Control (UI)	Default Value	Function
		Targets should still be specified as absolute paths in this node's parameters.
when enable orient constraint: enabled; orient constraint options		
targetPath	N/A	Specifies the object(s) to constrain to. If you want to aim a rig to point at a target, this is the target. If you set multiple targets, then the constraint aims at the average center of the objects. The **targetPath** parameter options are available in either a scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type in the Common Parameter Widgets.
addToConstraintList	yes	Specifies whether or not to add the base path for the rig to the globals.constraintList at **/root/world** in the **Attributes** tab.
targetOrientation	Object	Sets how the orientation of the target object is calculated: • **Object** - uses the local origin of the object as the target. • **Face** - uses the local origin on the face as the target.
allowMissingTargets	No	When set to **Yes**, silently ignore the constraint if its target is not in the scene graph. When set to **No**, produce an error on constraint resolution if the target is missing.
xAxis	Enabled	When enabled, orientation is constrained on the x axis.
yAxis	Enabled	When enabled, orientation is constrained on the y axis.
zAxis	Enabled	When enabled, orientation is constrained on the z axis.
setRelativeTargets	No	Stores target paths in the scene graph constraint definition as paths relative to the base path. Targets should still be specified as absolute paths in this node's parameters.

Control (UI)	Default Value	Function
Object list: light filter reference		
Object tab		
referencePath	N/A	The path to the light filter package that you want to reference as part of this light filter.

Control (UI)	Default Value	Function
Object list: light filter		
Object tab		
inherits	N/A	The path to another light filter package that you want to inherit from.
Object tab > viewer		
fill	solid	The manner in which the light filter is displayed in the viewer: as **points**, a **wireframe**, or a **solid** fill.
Object tab > transform		
translate	0.0, 0.0, 0.0	Transforms the light filter by moving it on the x, y, or z axes. Modifying the light filter in the viewer using the Translate manipulators also updates these parameter values.
rotate	0.0, 0.0, 0.0	Transforms the light filter by rotating it around the x, y, or z axes. Modifying the light filter in the viewer using the Rotate manipulators also updates these parameter values.
scale	1.0, 1.0, 1.0	Transforms the light filter by scaling it along the x, y, or z axes. Modifying the light filter in the viewer using the Scale manipulators also updates these parameter values.
Material tab		
useLookFileMaterial	disabled	When enabled, the material from an associated Look File is used.

Control (UI)	Default Value	Function
Add Shader	N/A	Click to add a renderer-specific shader from the dropdown list. The **Material** tab is populated with controls appropriate to the shader selected, and are dependent on the renderers installed.
useLookFileMaterial: enabled		
reference > asset	N/A	Set the path to the asset you want in your scene. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
reference > materialPath	N/A	Choose the material path from the dropdown list, based on the asset you have listed in the **asset** field above. If you do not have an **asset** listed in the asset field, nothing appears in the dropdown list for **materialPath**.
Linking tab > light linking		
action	append linking information	Determines whether the linking options set in this node override the incoming scene options or if the new settings are appended to the incoming options. If this light doesn't exist in the incoming scene, this option has no effect.
initialState	don't set value	Determines whether the newly-added light location is initially **on**, **off**, or doesn't have a value set.
clearUnmatched	disabled	When linking is resolved, the **clearUnmatched** parameter determines whether or not existing light linking attributes for this light are removed from locations that do not match the **on** or **off** expressions. The effect of this parameter is only visible in the **Attributes** tab when linking has been resolved, which means after a LightLinkResolve node or when Implicit Resolvers are active. Examines the **lightList** attribute on your linked objects to ensure that the attributes have been set correctly. If the parameter has been disabled, the value of the **enable** child attribute in the **lightList** attribute for your light is **0**; otherwise, the default enabled setting is **1**.

Control (UI)	Default Value	Function
on	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning the light **on** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to CEL Statement Widget Type in Common Parameter Widgets.
off	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning the light **off** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using the CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to CEL Statement Widget Type in Common Parameter Widgets.

Control (UI)	Default Value	Function
Object list: sky dome		
Object tab > transform		
translate	0.0, 0.0, 0.0	Transforms the sky dome light by moving it on the x, y, or z axes. Modifying the sky dome in the viewer using the Translate manipulators also updates these parameter values.

Control (UI)	Default Value	Function
rotate	0.0, 0.0, 0.0	Transforms the sky dome light by rotating it around the x, y, or z axes. Modifying the sky dome in the viewer using the Rotate manipulators also updates these parameter values.
scale	1000.0, 1000.0, 1000.0	Transforms the sky dome light by scaling it along the x, y, or z axes. Modifying the sky dome in the viewer using the Scale manipulators also updates these parameter values.
Material tab > material		
useLookFileMaterial	enabled	When enabled, the material from an associated Look File is used and the additional parameters are contained in the reference dropdown.
Add Shader	N/A	Click to add a renderer-specific shader from the dropdown list. The **Material** tab is populated with controls appropriate to the shader selected, and are dependent on the renderers installed.
useLookFileMaterial: enabled		
reference > asset	/tmp/hdriSkyDomeLight.klf	Set the path to the look file you've written to disk. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
reference > materialPath	/root/materials/hdriSkyDomeLight	Choose the material path from the dropdown list, based on the asset you have listed in the **asset** field above. If you do not have an **asset** listed in the asset field, nothing appears in the

Control (UI)	Default Value	Function
		dropdown list for **materialPath**.
Linking tab > light linking		
action	append linking information	Determines whether the linking options set in this node override the incoming scene options or if the new settings are appended to the incoming options. If this light doesn't exist in the incoming scene, this option has no effect.
initialState	don't set value	Determines whether the newly-added light location is initially **on**, **off**, or doesn't have a value set.
clearUnmatched	disabled	When linking is resolved, the **clearUnmatched** parameter determines whether or not existing light linking attributes for this light are removed from locations that do not match the **on** or **off** expressions. The effect of this parameter is only visible in the **Attributes** tab when linking has been resolved, which means after a LightLinkResolve node or when Implicit Resolvers are active. Examines the **lightList** attribute on your linked objects to ensure that the attributes have been set correctly. If the parameter has been disabled, the value of the **enable** child attribute in the **lightList** attribute for your light is **0**; otherwise, the default enabled setting is **1**.
on	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning the light **on**

Control (UI)	Default Value	Function
		for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to CEL Statement Widget Type in Common Parameter Widgets.
off	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning the light **off** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using the CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to CEL Statement Widget Type in Common Parameter Widgets.
Linking tab > shadow linking Support for shadow linking is dependent on your renderer.		
action	append linking information	Determines whether the linking options set in this node override the incoming scene options or if the new settings are

Control (UI)	Default Value	Function
		appended to the incoming options. If this light doesn't exist in the incoming scene, this option has no effect.
initialState	don't set value (or use existing value for adopted lights)	Determines the initial value for shadow visibility in the **lightList** entry for the newly-added light: **on**, **off**, **don't set value**, or **use existing value**.  **Note:** The **use existing value** option is only available for adopted lights.
clearUnmatched	disabled	When linking is resolved, the **clearUnmatched** parameter determines whether or not existing light linking attributes for this light are removed from locations that do not match the **on** or **off** expressions. The effect of this parameter is only visible in the **Attributes** tab when linking has been resolved, which means after a LightLinkResolve node or when Implicit Resolvers are active. Examines the **lightList** attribute on your linked objects to ensure that the attributes have been set correctly. If the parameter has been disabled, the value of the **enable** child attribute in the **lightList** attribute for your light is **0**; otherwise, the default enabled setting is **1**.
on	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning shadow

Control (UI)	Default Value	Function
		casting from this light **on** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to the CEL Statement Widget Type in Common Parameter Widgets.
off	N/A	A CEL Statement through which scene graph locations can be linked to the selected light, thereby turning shadow casting from this light **off** for those locations. The scene graph locations are specified using the Collection Expression Language (CEL). For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**). CEL Statements are edited using CEL Statement Widgets. For more information on the CEL Statement Widget type, refer to the CEL Statement Widget Type in Common Parameter Widgets.

ImageCoordinate

ImageCoordinate allows you to load an image into the interface and specify a 2D-point (x, y coordinates) that is then stored as attribute data on a scene graph location.

Control (UI)	Default Value	Function
location	/root/world/2d_image/points	The scene graph location to where (x, y) coordinates are written. The **location** parameter options are available in either the scene graph widget or ▼ dropdown menu to the right of the parameter. For more information, refer to the Create Scene Graph Location Widget Type in the Common Parameter Widgets.
filePath	N/A	Specifies the file path to the image to load here. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
numberValue	1x2	Sets the override value.

PonyStack

The PonyStack node is an example node that allows you to create multiple pony geometry locations with differing transform attributes.

Control (UI)	Default Value	Function
location	/root/world/geo	Specifies the ponies scene graph location. For more information, refer to the Create Scene Graph Location Widget Type in the Common Parameter Widgets.

Control (UI)	Default Value	Function
➕	N/A	Creates a new pony in the object list.
transform		
transform	N/A	Transforms each of the ponies according to the Scale, Rotation, and Translation (SRT), or matrix controls. For more information, refer to the Transform Controls Widget Type in the Common Parameter Widgets.

Other Nodes (Misc)

The following section describes Katana's **Other** miscellaneous nodes.

Backdrop

The Backdrop node allows you to improve readability and navigation of your recipes. For example, you can set a colored background around a group of nodes, or use the Backdrop node to add comments in a scene. For more details on the Backdrop node, see Backdrop Nodes

DependencyMerge

The DependencyMerge node takes any number of Render dependencies as inputs and consolidates them into a single link that you can wire into your dependent Render node.



Note: Add as many ports as necessary by clicking the arrow at the top of the node in the **Node Graph** tab.

Dependencies between Render nodes are represented by links between the nodes. This can rapidly become very complex, since a single Render node may depend on several other Render nodes throughout the node graph.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	The render discrepancies that you want to consolidate.

Control (UI)	Default Value	Function
farmSettings		
farmFileName	N/A	Sets the location where the farm file is written.

Dot

The Dot node performs no operation on the data passing through it. Its purpose is to improve the appearance and layout of your node graph, but also to disable connections between nodes. In this way, you can use the Dot node as an on/off switch for incoming connections.



Tip: You can insert Dot nodes on-the-fly during link creation by pressing the **.** (**period**) key.

Connection Type	Connection Name	Function
Input	input	The place in the node graph after which you want to create a bend in the arrow.

Control (UI)	Default Value	Function
Display As Dot	enabled	When enabled, Dot nodes are displayed as a dot in the node graph instead of the regular rectangle node shape.

Group

The Group node is a node that contains other nodes. You can create Group nodes by selecting some nodes and pressing **G** to collapse them into a Group. To do the inverse (explode a Group), select a Group node and press **U**.

You can convert a Group node to a LiveGroup by right-clicking on the Group node and selecting **Convert to LiveGroup**.

GroupStack

The GroupStack node is a SuperTool that creates a convenient interface for managing a list of nodes of the same type.

Within the GroupStack interface, you can create any number of **nodes of the same type**, and these nodes are linked together, providing a single output by connecting them one after the other in serial, in the order in which they appear in the stack. GroupStack is similar to GroupMerge, except with GroupMerge the nodes are merged together instead of creating a list of nodes where a 3D input is passed through.

This node is most often used to group nodes that have one input and modify the scene graph in some way. For example, the GroupStack node could be used to manage multiple CollectionCreate nodes, or multiple material edits.



Note: When the GroupStack node is first created, its type is not defined. You can create a node and then add it to the stacklist by **Shift**+middle-mouse and dragging from the **Node Graph** tab to the node's list in the **Parameters** tab. At that point, the GroupStack is permanently typed as a group of the type of node that was dragged in.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to create a SuperTool for managing several nodes of the same type.

Control (UI)	Default Value	Function
	N/A	Creates a new node of the type associated with this node and adds it to the node list.
	N/A	Brings up a searchable list to aid in selection.
/	N/A	Locks all nodes against editing. Unlocks all nodes for editing.
[Right-click menu]		
Ignore Selected Entries	N/A	Disables the selected nodes.
 View At Location	N/A	Sets the current view node to the selected node
Delete Selected Entries	N/A	Deletes the selected node.
Duplicate Selected Entries	N/A	Duplicates the selected node, creating a new copy of both the node and matching its parameters.
Cut Selected Entries	N/A	Deletes the selected node and copies it to the clipboard.
Copy Selected Entries	N/A	Copies the selected node to the clipboard.
Paste	N/A	Paste the current clipboard node into this list.
Tearoff Parameters Of Selected Entries...	N/A	Create a new floating window with the parameters of this node on a tab inside.

InteractiveRenderFilters

Interactive render filters enable you to setup common interactive render recipe changes without having to include them within the recipe. These filters are designed to only be included when performing an interactive render and are ignored for Disk Renders.

InteractiveRenderFilters nodes don't need to be connected into a recipe to take affect.

An example of a render filter is a render resolution change. You can set up an interactive render filter to reduce the size of a render, thus making debugging and light tests much quicker. Other examples might be changing anti-aliasing settings or the number of light bounces.

LiveGroup

The LiveGroup node is similar to the Group node except the contents are loaded from an external file. The contents of the LiveGroup can be locked (non-editable) or unlocked, and is used mainly during look development. To change the contents of a LiveGroup, you can modify the file it references or change the contents of a LiveGroup in a Katana session, then publish it back to the source file. The LiveGroup context menu is dynamic, and the options that appear in the menu change depending whether the node is in a non-editable or editable state.



Note: The source file is automatically reloaded each time a scene is opened in Katana. If the file has been changed, the changes are picked up automatically. If the source file cannot be read or no longer exists, a copy stored in the scene file is used instead, and a warning is printed to the shell.

Control (UI)	Default Value	Function
source	N/A	Sets the path to load in the source (file) as the contents of the LiveGroup. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.

Control (UI)	Default Value	Function
	N/A	Opens the **Parameters** tab.
[Right-click menu] (when non-editable)		
Load...	N/A	Opens the **Load LiveGroup** dialog to allow you to select a LiveGroup source for the node.
Revert	N/A	Reloads the LiveGroup from its current **source**, thus discarding any changes made while it was unlocked. **Revert** is only available when a **source** has been set.
Edit Contents	N/A	Changes the state of the node to be editable, so that contents of the node can be modified.
Convert to Group	N/A	Converts the LiveGroup node to a Group node.
Other | Show Parameters	N/A	Opens the **Parameters** tab.
[Right-click menu] (when non-editable)		
Load...	N/A	Opens the **Load LiveGroup** dialog to allow you to select a LiveGroup source for the node.
Revert	N/A	Reloads the LiveGroup from its current **source**, thus discarding any changes made while it was unlocked. **Revert** is only available when a **source** has been set.
Publish...	N/A	Opens the **Publish LiveGroup** dialog for publishing the parameters and contents of the LiveGroup node as a LiveGroup source file or asset. This leaves the node in an editable state for further changes.
Publish and Load...	N/A	Opens the **Publish and Load 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 sets the node to be non-editable and locks the node against further changes.
Convert to Group	N/A	Converts the LiveGroup node to a Group node.
Other | Show Parameters	N/A	Opens the **Parameters** tab.

LookFileAssign

Assigns a Look File to a scene graph location defined by a CEL statement.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to assign a look file to the scene graph location.

Control (UI)	Default Value	Function
CEL	None	Specifies the scene graph location(s) where the Look File is assigned. The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
asset	None	The Look File that is assigned to the specified scene graph location(s). For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.

LookFileGlobalsAssign

LookFileGlobalsAssign nodes associate a look file with the **/root** location and is designed to repeat the changes made to that location (LookFiles for assets are assigned to the location of the asset).

Also see LookFileManager.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to assign a look file to the **/root** location.

Control (UI)	Default Value	Function
asset	N/A	The asset to assign to the Look File. For more information, refer to the Asset and File Path Widget Types in the Common Parameter Widgets.
resolveImmediately	No	When set to **Yes**, LookFileResolve runs on the root of the scene as part of this node. This is useful for overriding or layering scene root attributes from published Look File assets This option has special behavior during Look File baking. Instead of resolving, it appends the Look to the lookfile.referencedAssets attribute. This gets included in the resulting Look File and maintains a live reference to it during subsequent Katana standard resolution.
Flush Look File Cache	N/A	Click to flush the Look File cache and force a reload.

MaterialAssign

Assigns materials to geometry in the scene graph.

Connection Type	Connection Name	Function
Input	input	The material in the node graph that you want to apply tot he geometry in the scene graph.

Control (UI)	Default Value	Function
CEL	N/A	Sets the CEL specification of scene graph locations on which the assignment acts. For more information, see the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL widget parameters in Common Parameter Widgets.
materialAssign	N/A	Specifies the material to assign. Typically, you middle-mouse drag this from under **/root/materials/geo** in the scene graph. The **materialAssign** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Path Selection Widget Types in Common Parameter Widgets.

NonpersistentSwitch

This node is identical to the Switch node, except that the in control is reset to 0 whenever the file is loaded (the value you set it to in your current session is never saved to the **.katana** file).

This is useful for switches you may want to use interactively (low-quality settings, for example), that you don't want to mistakenly have set for a batch render. Using a NonpersistentSwitch ensures that batch renders always get the left-most input to the node.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	The input ports you want to set for different parts of the node graph.

RenderScript

This node generates a user-specified command in the outline script, following the same dependency rules as the Render node. This node is not renderable interactively or using the **--batch** command.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	The input ports you want to set for different parts of the node graph.

Control (UI)	Default Value	Function
command	shell	Sets the outline function to generate. Default value of 'shell' expects a single commandArgument, which is the shell command to run on the farm. Example: command("nodeName", "commandArg1", "commandArg2", "keywordName1" => "keywordValue1", "keywordName2" => "keywordValue2",)
commandArguments		
commandArguments	N/A	Array of positional arguments added to the outline function.

Control (UI)	Default Value	Function
		Example: command("nodeName", "commandArg1", "commandArg2", "keywordName1" => "keywordValue1", "keywordName2" => "keywordValue2",)
keywordArguments		
keywordArguments	N/A	Array of keyword arguments (name => value pairs) added to the outline function. Example: command("nodeName", "commandArg1", "commandArg2", "keywordName1" => "keywordValue1", "keywordName2" => "keywordValue2",)
pythonImports		
pythonImports	N/A	Array of import statements to be added to the Python farm file. Example: from outline.modules.shell import Shell or

Control (UI)	Default Value	Function
		import outline.module.shell
farmSettings		
setActiveFrameRange	disabled	When enabled, activeFrameRange parameters are exposed to define the active frame range for rendering. When disabled, the active frame range is assumed to be the same as globals.inTime and globals.outTime.
setActiveFrameRange: enabled		
start	1	When **setActiveFrameRange** is enabled, sets the start of the active frame range.
end	1	When **setActiveFrameRange** is enabled, sets the end of the active frame range.
farmSettings continued		
dependAll	disabled	When enabled, farm dependencies wait until all frames of this node are rendered before rendering themselves.
threadable	enabled	
farmFileName		
excludeFromFarmOutputGeneration	disabled	When enabled, this node does not appear in any generated farm file (however, the node is still renderable if called directly).
forceFarmOutputGeneration	disabled	When enabled, this node always appears in a generated farm file (regardless of whether it has any valid outputs).

Control (UI)	Default Value	Function
		Note: If **excludeFromFarmOutputGeneration** is also set, the node does not appear in the generated farm file (**excludeFromFarmOutputGeneration** overrides **forceFarmOutputGeneration**).
inputs		
inputs		

RenderSettings

The RenderSettings node defines the 3D render output settings (camera to use, renderer, size of output) for an image.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to define the 3D render output settings.

Control (UI)	Default Value	Function
cameraName	/root/world/cam/camera	Specifies the camera that the scene should be rendered through. The field contains a path to the camera's location in the scene graph. The **cameraName** parameter options are available by clicking the dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common

Control (UI)	Default Value	Function
		Parameter Widgets.
renderer	dl	Specifies the renderer to use.
resolution	512x512	Sets the size of the output image.
overscan	0	Pads the data window of the resulting render by the specified pixel amount on each side. The frame window is unchanged.
adjustScreenWindow	No adjustment	Adjusts the pixel aspect ratio to match one of the device aspect ratio's dimensions. Either the height or the width of the screen window is adjusted to match the output resolution.
maxTimeSamples	1	Sets how many times a point is sampled when the shutter is open. For animated parameters within Katana (such as transforms), this is how many samples are evaluated from shutter open to close. The higher the number, the more accurate the motion blur.
shutterOpen	0	Specifies the timing of the opening and closing of the camera shutter.
shutterClose	0	
cropWindow	0.0, 1.0, 0.0, 1.0	Specifies the render crop window in normalized coordinates: xmin xmax ymin ymax, starting in the upper left-hand corner. The part of the image that renders has a dotted red line around it. **Note:** The dotted red line isn't displayed unless you are viewing the RenderSettings node.
interactiveOutputs	all	Specifies whether **all** the AOVs are

Control (UI)	Default Value	Function
		rendered during Preview Render, or whether only the **primary** pass is rendered. If you set the output to **primary**, the Local Assignment box turns yellow to indicate that you are using local values.
interactivePrecision	float	The data type to use for encoding pixels that are transmitted between a render process and Katana. You can use this to reduce the bandwidth required for transmitting images, but at the cost of a loss of fidelity. Choose between three options: • **float** : Single-precision floating-point format (32 bits). • **half** : Half-precision floating-point format (16 bits). • **byte** : Reduced precision resulting in the lowest bandwidth, but possibly visible banding artifacts and a loss of fidelity when stopping images up or down (8 bits). This clamps pixel values to a range between 0 and 1. The default value can be set using the KATANA_DISPLAYDRIVER_PRECISION environment variable.
geolibRuntime	Geolib3-MT	Set which Geolib3 Runtime to use. In the majority of cases, you should select **Geolib3-MT**. However, a small number of scenes that are not set up for concurrent scene traversal may perform better with the **Classic** Runtime from Katana 3.2. See Geolib3-MT sceneTraversal Parameters

Control (UI)	Default Value	Function
		for parameter details.
cacheEvictionMode	DependencyProtecting	Choose a cache eviction strategy from one of the three options: • **Dependency protecting** • **Continual** • **Relaxed** For details see Performance Optimization using a Cache Eviction Strategy.
deferredUsdToKatanaEnabled	on	Toggles the conversion of USD data to Katana data when performing a render or with Implicit Resolvers enabled.
adjustScreenWindowWhen	deferred	Determines how the adjustScreenWindow parameter (see above) is handled: • **deferred** - adjustScreenWindow is evaluated by the AdjustScreenWindowResolve Op as an implicit resolver. • **immediate** - adjustScreenWindow parameter takes effect immediately.

Geolib3-MT sceneTraversal Parameters

opTreeOptimizations checkbox Default: Off	When turned on, Geolib3-MT performs a pre-processing step in which it examines the topology of the Op tree to identify constructs that can be potentially optimized. One optimization is the collapsing of sequences of Ops of the same type into a single instance of that Op. There are a number of benefits to this:

	• Reduced function call overhead - There is a small cost involved in scheduling an Op to cook a scene graph location. By combining chains of similar Ops, it's possible to reduce this function call overhead. • Reduced memory footprint - A chain of 10 Ops occupies 10 separate cook results in the caching subsystem, while a successfully collapsed Op Chain occupies only 1 cook result per location. The result of evaluating a collapsed chain of Ops, when observed from the most downstream Op, should be the same as if the chain of Ops were evaluated.  **Note:** Op API calls to query upstream scene graph results, such as getAttr(), does not return the expected result when called within a collapsed chain if one of those Ops within the chain was responsible for setting that attribute. In this case the Op should use getOutputAttr() instead. The Op tree optimizer attempts to collapse any chain of Ops of the same type if it calls **GeolibSetupInterface::setOpsCollapsible()** during the **setup()** call. Callers of this function must specify the name of an attribute which Geolib3 passes to the Op's **cook()** call as an Op argument. This attribute contains an ordered array of attributes (ordered upstream Op to downstream Op) containing the collapsed Ops' arguments. The Op is then able to deal with this batch Op argument as appropriate.
verboseLogging checkbox Default: Off	When turned on, verbose logging inside the Geolib3-MT runtime is enabled. This includes verbose logging during scene traversal, and Op tree optimization, if enabled.

sceneTraversal.cache Parameters

Each Geolib3-MT cache eviction strategy, listed under the **cacheEvictionMode** parameter, includes a number of configurable settings based on which eviction strategy you choose. These settings can be modified from the **sceneTraversal.cache** menu and can be optimized on a project-by-project basis.

Caching, and the trade-off between memory usage and time to first pixel can have a significant impact on the performance of scene traversal time and rendering. Using the settings provided it's possible to tune the memory footprint during the scene traversal phase of rendering.

cacheEviction checkbox Default: On	When enabled, data in the cache will be periodically removed (evicted) based on the selected **cacheEvictionMode**. **Note:** The Geolib3-MT Runtime maintains a cache to improve the locality of previously cooked data. Cache entries are keyed on pairs of scene graph location and Op ID. Disabling cache eviction can decrease scene graph traversal times but may lead to higher memory usage. When cacheEviction is disabled with large scene graphs, the available memory can quickly become full. **Tip:** While initially it might seem counter-intuitive to disable cache eviction, there may be scenes where it is appropriate. This may be the case when the scene and data structures required by the renderer fit comfortably into memory. Even larger scenes could benefit to some degree, as once the scene generation phase of rendering is complete, the memory pages occupied by Geolib3-MT's cook results can no longer be accessed and therefore are not eligible for paging to disk; as these pages won't be re-paged to main memory during rendering the performance penalty is minimal.

Dependency Protecting sceneTraversal.cache Parameters

evictionTriggerTimeInterval Default: 1000ms	The time interval for the time-based scene data cache eviction trigger is specified in milliseconds. 0ms disables time-based eviction.
evictionTriggerCookCount Default: 0	The cook count for which the cook-based scene data cache eviction is triggered. 0 disables cook-based eviction.
mruCookProtectionCount Default: 0	Specify the maximum number of recently used cooks to safeguard scene data from being evicted.

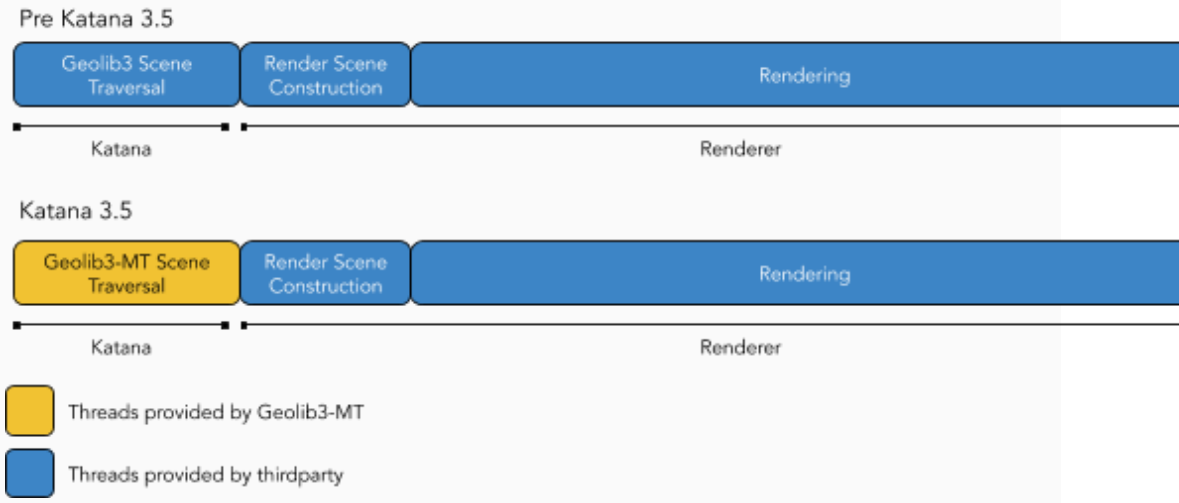
Continual sceneTraversal.cache Parameters

cacheLimitingMode Default: **Fixed cache size**	Specifies the method for cache eviction. • **Fixed cache size** : If the cache size reaches a set limit of the number of entries, every new entry triggers the removal of one of the least-recently used. • **Memory-based** : If the used memory in the system reaches a set level, each new cache entry causes removal of a set number of the least-recently used entries. For **Memory-based**, you can set three levels of memory usage: light, medium, and heavy, to increase the amount of memory used for the cache. Each level has an eviction count to set how many entries are evicted when the cache hits that level.
cacheLimitingMode: Fixed cache size	
cacheSize Default: 1000000	Determines the set size of the cache. Each entry in the cache is identified by a combination of a scene graph location and an Op ID. When the cache limit is reached, every new cache entry triggers the removal of one of the least-recently used. A smaller cache size uses less memory, but it may lead to recooking of evicted data if it gets removed prematurely, leading to reduced performance.
cacheLimitingMode: Memory-based	
lightMemoryLevel Default: 70% **lightEvictionCount** Default: 1	Set the lowest percentage level of memory usage at which new entries will trigger the removal of least-recently used entries. The number removed is set by **lightEvictionCount**
mediumMemoryLevel Default: 80% **mediumEvictionCount**	Set the middle percentage level of memory usage at which new entries will trigger the removal of least-recently used entries. The number removed is set by **mediumEvictionCount**

Default: 2	
heavyMemoryLevel Default: 90% **heavyEvictionCount** Default: 3	Set the middle percentage level of memory usage at which new entries will trigger the removal of least-recently used entries. The number removed is set by **heavyEvictionCount**

Relaxed sceneTraversal.cache Parameters

cacheSoftLimit text field Default: 1,048,576	If **cacheEviction** is on, the **cacheSoftLimit** governs how many cook results are stored in local caches before entries are evicted using a least recently used eviction policy.  **Note:** Whilst these entries may be evicted from a local cache they may be shared amongst a number of other local caches or the central (shared cache). In which case, the entries' memory won't immediately be reclaimed. Consider the maximum depth of the scene graph and Op tree. The **cacheSoftLimit** controls the size of the recently used cook result cache on a per thread basis. This means any locations cooked on a particular thread, or any locations accessed during the cooking process (such as via **getAttr()**), are stored in the local cache and subject to eviction based on the value of the **cacheSoftLimit**.
collectionFrequency text field Default: 10,000	If **cacheEviction** is turned on, the **collectionFrequency** governs the time, in milliseconds, between collection cycles. During a collection cycle, Geolib3-MT gathers all cache entries evicted since the previous collection cycle and if the cook result is no longer used, evict and reclaim the memory for the cook result.

	Note: Reducing the collectionFrequency interval causes more aggressive eviction of cook results leading to a reduced memory footprint but potentially at the cost of scene traversal time.
useCachePrepopulation checkbox Default: On	If turned on, Geolib3-MT performs a traversal of the scene graph populating an internal cache. The extent of this traversal can be controlled by the settings under **sceneTraversal.cachePrepopulation**.
maxCores text field Default: 0	Determines how many logical cores Geolib3-MT uses during scene traversal phase. Unlike previous versions, Geolib3-MT uses an internal thread pool to improve scene traversal time. The following diagram demonstrates the difference between Geolib3-MT and previous versions of Katana.  The default value of 0, causes Geolib3-MT to use all available logical cores on the host computer.



Note: Whilst the core Geolib3-MT processing engine scales as the number of cores increases, individual Ops within an Op tree may not exhibit the same scaling characteristics. It is possible that an increase in threads can result in an increase of scene traversal time. In this case, the new profiling tools available in Katana 3.5 can be used to identify these Ops and optimize their behavior. The same is true of Ops marked **thread unsafe**, as these require the acquisition of a **Global Execution Lock (GEL)**, which further limits scene traversal scalability.

Relaxed sceneTraversal.cachePrepopulation Parameters

preCookSourceOps checkbox Default: Off	If turned on, Geolib3-MT first fully traverses the scene generated by any source Op (any Op with no inputs) found in the Op tree. This setting can provide benefits when loading in geometry caches or other asset types.  **Note:** Empirical tests have found that source Ops are typically followed by some form of prune operation; as a result, in these cases, turning on preCookSourceOps can generate more scene graph locations than is required which can lead to increased memory consumption and traversal times.
preCookKeyOps checkbox Default: On	If turned on, Geolib3-MT identifies Ops within the Op tree that can be evaluated in parallel. An example of this is the **Merge** Op: Geolib3-MT evaluates each branch in parallel, which can reduce scene traversal times.
preCookAllLocations checkbox Default: On	If turned on, Geolib3-MT cooks all remaining scene graph locations, fully expanding the scene.

Based on the values of the settings above, on completion of the **cachePrepopulation** phase, the Geolib3-MT cache is pre-populated with either the whole scene graph or a subsection of it. Geolib3-MT has been optimized to provide efficient access to renderer plugins via the existing **FnScenegraphIterator** API to this

cache. This cache is a scalable, thread safe cache, as such >we encourage renderer plugins to access this cache concurrently to improve the performance of the scene build phase.



Warning: If the Geolib3-MT cache is not fully populated, cache access (via FnScenegraphIterator) results in a cache miss. In this case the requested location is cooked using the calling thread.

deferredUsdToKatana Settings

These settings determine how the USD data is converted to Geolib data before being sent to a renderer.

They are the same settings found in UsdToKatana but in this case, the conversion is deferred to render time.

RendererProceduralAssign

The RendererProceduralAssign node allows you to assign renderer procedurals to specific locations in the scene.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to assign renderer procedurals.

Control (UI)	Default Value	Function
CEL	N/A	Sets the CEL specification of scene graph locations on which the assignment acts. The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
rendererProceduralAssign	N/A	Specifies which procedural to assign. Typically, you'll middle-mouse drag this from under **/root/materials/proc** in the scene graph. The **rendererProceduralAssign** parameter options are available by clicking the ▼ dropdown menu. For more information, refer to the Scene Graph Location Widget Type in Common Parameter Widgets.

ScenegraphObjectSettings

The ScenegraphObjectSettings allows you to add attributes, used by the **Scene Graph** tab, to locations according to the CEL match.

Connection Type	Connection Name	Function
Input	input	The place in the node graph where you want to add attributes to specified locations.

Control (UI)	Default Value	Function
CEL	N/A	Specifies the scene graph location(s). The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
stopExpand	No	The **Scene Graph** tab "Expand..." actions do not traverse beneath locations at which this enabled.

Switch

This node allows you to switch between multiple input nodes. Only the portion of the node graph connected to the selected input port on the Switch node is evaluated. You can select which input to choose using the **in** parameter on the node.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	The input ports you want to set for different parts of the node graph.

Control (UI)	Default Value	Function
in	0	Selects the index of the input port (starting at zero) to pass through to the output.

Teleport

This node can be used to visually clean up a scene by hiding the lines between nodes. In order to attach more than one node to the Teleport node, click **Add** > **Add Pass Input**. Then, in the Node Graph, you can drag a line from any node to connect it. When inputs are not shown, each connected node appears on the Teleport node as an output arrow.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to clean up the scene by hiding connecting lines between nodes.

Control (UI)	Default Value	Function
name	output	Sets the name for each input.
Show inputs	disabled	When enabled, the connector between this node and the inputs are not shown

TimeOffset

In Katana the current time used in parameter evaluation is a property that flows up the graph and is referenced as a frame in parameter expression. This node modifies that time in upstream nodes. Common uses are to offset or lock data loaded from an upstream input.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to modify the time of upstream nodes.

Control (UI)	Default Value	Function
inputFrame	The value of **frame**: a Python-based parameter expression that evaluates to the current frame.	Sets the value of frame in input nodes. This can be an expression, for example frame + 10.

UsdIn

The UsdIn node allows you to import USD assets into Katana.

Universal Scene Descriptor (USD), is an open-source scene information interchange framework developed by Pixar. USD allows you to assemble and organize large amounts of assets into scenes or shots and transfer them between Digital Content Creation packages in a non-destructive way. USD files can contain geometry, materials and shading assignments, lighting, and physics.

For more information on USD, see https://graphics.pixar.com/usd/release/intro.html



Note: For versions of Katana before 4.5v1, you must to enable the USD plug-in using environment variables. For more information on how to do this, see Loading USD Plug-ins into Katana.

Control (UI)	Default Value	Function
fileName	N/A	Specifies where to retrieve the asset, a USD (**.usd**, **.usda**, **.usdc**, or **.usdz**) file. For more information, refer to Asset and File Path Widget Types.
assetResolverContexts	N/A	Sets the Asset Resolver Context which will be bound when opening the stage. If this is not specified, a default context is used based on the Asset Resolver for the fileName parameter.

Control (UI)	Default Value	Function
location	/root/world/geo/asset	Specifies the scenegraph location where the USD asset is to be placed. The location parameter options are available in either the scenegraph widget or dropdown menu to the right of the parameter. For more information, refer to the Scene Graph Location Widget Type.
isolatePaths	N/A	Used to load only the USD contents below a specified USD prim path.
variants (deprecated)	N/A	Used to specify USD Variants. This parameter is deprecated, we recommend to using USDVarientSelect instead.
ignoreLayerRegex	N/A	Ignores matching USD layers when a scene is being loaded into Katana.
motionSampleTimes	N/A	Specifies the frame-relative sample times to use when motion blur is requested using two input frames separated by a space. For example: Entering **0 1** loads all samples that contribute to the current frame's shutter window. Entering **-1 1** loads all samples that contribute to the previous, current, and next frame's shutter window. The default behavior is no motion samples, load only the current frame. Generally, using more samples produces smoother motion blur, but at the cost of longer render times.

Control (UI)	Default Value	Function
		The number of **motionSampleTimes** can be overridden with either UsdInDefaultMotionSamples or UsdInMotionOverrides nodes downstream. **Note:** The RenderSettings node **maxTimeSamples** control must be set to match the second sample time input to enable motion blur in renders.
instanceMode	expanded	When set to **expanded**, instances are loaded as though the children of parent primitives were there naturally. When set to **sources and instances**, parents of instanced primitives are created under a sibling of **/world** named **/instances**.
usdPurposeBasedMaterialBinding	No	When enabled, allows you to specify material bindings to be loaded in for objects with a purpose tag within the USD file. When disabled, Katana will load in the full USD scene with all material bindings resolved. **Note:** For more information, see USD Glossary: Purposes.
prePopulate	enabled	Assumes all assemblies and pre-populates present in your USD file are

Control (UI)	Default Value	Function
		needed within the scene and loads them all in. Having prePopulate is good for efficiency, as it allows USD to use its own multithreading.
verbose	disabled	When enabled, the information generated during USD scenegraph generation is output to the command line.
asArchive	disabled	When enabled, the type specified in the location parameter is **usd archive** instead of loaded directly into the scene. If enabled, **variants**, **ignoreLayerRegex**, **motionSampleTimes**, and **instanceMode** are made unavailable.
evaluateUsdSkelBindings	enabled	

UsdInActivationSet

Activates or deactivates scopes within the USD stage using the session layer. Deactivated scopes (and their children) aren't include in stage composition, so it's often more efficient to deactivate something unused there instead of Katana's Prune or Isolate nodes. Katana doesn't read USD data from locations that are removed by Prune or Isolate nodes, but they are still considered during USD stage composition.

For more information on USD, see https://openusd.org/release/intro.html

Control (UI)	Default Value	Function
locations	None	One or more Katana scenegraph locations loaded from a UsdIn node.
active	No	Sets whether the scope is activated, **Yes**, or de-activated, **No**.

VariableDelete

This node deletes the locally-set graph state variable listed in the node's **variableName** parameter.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to delete locally-set graph state variables.

Control (UI)	Default Value	Function
variableName	var1	Specify the name of the local graph state variable that is to be deleted by the node.

VariableEnabledGroup

This node works like a group, but allows you to enable or disable a variable pattern to which other variable nodes attempt to match.

Control (UI)	Default Value	Function
variableName	var1	Specify the name of the graph state variable to which you want to match a specific pattern.
pattern	N/A	Specify the pattern input that the graph state variable should match in order to enable the variable group. A pattern can be any input. The **pattern** parameter is a CEL statement widget. For more information about CEL statements, refer to Common Parameter Widgets.

VariableSet

This node sets a graph state variable pattern to which other variable nodes attempt to match.

Connection Type	Connection Name	Function
Input	in	The place in the node graph where you want to set a graph state variable pattern.

Control (UI)	Default Value	Function
variableName	var1	Specify the name of the graph state variable to which you want to match a specific pattern.
variableValue	N/A	Specify the value input that the graph state variable should match. A value must be a string.

VariableSwitch

This is a specialized Switch node that selects an input based on the value of a Graph State Variable.

Connection Type	Connection Name	Function
Input	Add numbered input ports (i0, i1, i2) by pressing ▼ in the node.	The input ports you want to set for different parts of the node graph.

Control (UI)	Default Value	Function
variableName	var1	The name of the Graph State Variable used to select an input port.
patterns	N/A	A set of patterns linked to input ports, used to select an input port according to the value of the Graph State Variable. Patterns may be added and deleted using the wrench dropdown menu for the group and existing patterns, respectively. Input ports for which no pattern is defined match their exact port name. 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.

Note: If a VariableSwitch node defines no patterns, input select is performed using a faster look-up operation. This may be useful for nodes with a large number of input ports.

ViewerObjectSettings

Adjusts how objects are displayed in the **Viewer** tab.

Connection Type	Connection Name	Function
Input	in	The object whose display you want to modify in the **Viewer** tab.

Control (UI)	Default Value	Function
CEL	N/A	The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add**

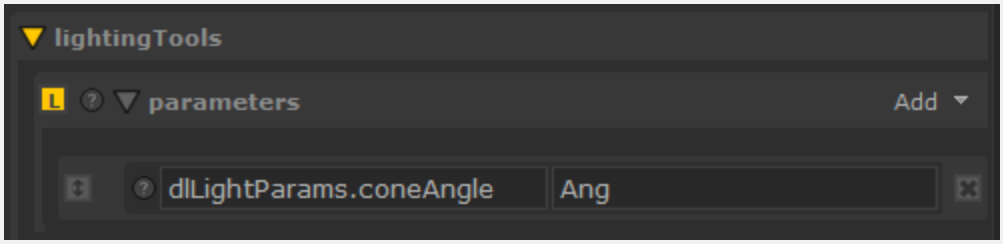
Control (UI)	Default Value	Function
		Statements. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
drawOptions		
hide	No	Sets whether the object should be hidden in the Viewer.
fill	inherit	Sets how the object is displayed, as: • **points** - display the object using points at the vertices or control points. • **wireframe** - display the object using wireframe mode. • **solid** - display the object as a solid. If the display style for the object uses a 3D lighting model, then display the object using that lighting model, whereas if the **Viewer** tab's display style is points or wireframe, display the object using a single solid color. • **inherit** - no change to the object's display style, use the default.
light	inherit	Sets the lighting model for the object. This setting doesn't influence the object when it is drawn using wireframe or points. You can set it to: • **default** - uses the simple shaded lighting model. • **shaded** - uses the viewer shader assigned to the object (or the default viewer shader if one isn't assigned). • **inherit** - don't override the **Viewer** tab display style.
smoothing	inherit	When the objects referenced by the CEL statement are being displayed as points or lines, this parameter sets whether they should be anti-aliased. The options are: • **off** - no anti-aliasing.

Control (UI)	Default Value	Function
		• **lines** - when displayed as a wireframe, the objects are anti-aliased. • **points** - when displayed using points, the objects are anti-aliased. • **both** - when displayed as a wireframe or using points, the objects are anti-aliased. • **inherit** - no object specific override, use the current default.
windingOrder	inherit	Sets whether the object has a clockwise or counterclockwise winding order. The winding order determines which direction is considered out from an object and which direction is in.
pointSize	4	Sets the size of the points when the object is rendered as a series of points.
color	0.4, 0.4, 0.4	Sets the default draw color. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
faceCulling	inherit	Allows geometry in the Hydra Viewer to be rendered without culling faces. • **back** - cull back-facing faces • **front** - cull front-facing faces • **none** - don't cull any faces • **inherit** - inherit the culling style from ancestor locations By default, back-facing faces are culled in the Hydra Viewer. **Note:** The OSG Viewer does not cull faces.

Control (UI)	Default Value	Function
annotation		
text	None	Sets the text to display with the geometry. When empty, no tag is displayed.
color	0.4, 0.4, 0.4	Sets the default background color for any annotation text. For more information, refer to the Color Widget Type in the Common Parameter Widgets.
ViewerObjectSettings parameters continued		
pickable	Yes	Sets whether the object is pickable or not.
resolveMaterialInViewer	default	Controls whether the **Viewer** should resolve materials. • **default** - Use default rules to resolve materials in the **Viewer**. • **always** - Always resolve materials in the **Viewer**. • **never** - Never resolve materials in the **Viewer**.

Lighting Tools

parameters	N/A	Add extra parameters to be displayed in the Lighting Tools light parameter widgets when a light is selected, provided the parameters are present in the light. The parameters are added after the default parameters **Intensity**, **Exposure** and **Color**. Enter the parameter path in the first text field, and the label for your widget parameter to use in the second text field. • Parameter path For example, **dlLightParams.coneAngle** • Label (optional) For example, **Ang**

		 **Note:** If the label is not provided, the parameter's original label will be used. **Note:** For a workflow example on customizing your light parameter widgets, see Customizing the Light Widget.
hideHandle	No	Determines whether the center of interest arrow handles are drawn for unselected lights. • **Yes** - Center of interest arrow handles are hidden for unselected lights. • **No** - Center of interest arrow handles are not hidden for unselected lights.

VisibilityAssign

The VisibilityAssign node changes the visibility setting of objects in the scene. The attribute is inherited, thus large sections of the scene graph can be made visible/invisible by assigning to common parents

A child can be explicitly set to visible even if its parent is not visible. For example, to render just one of several siblings, set the parent's visibility to 0, and set the item to render's visibility to 1. All siblings that are not explicitly marked picks up the parent's visibility setting of 0, but the item to render uses its explicitly set value of 1.

The **Scene Graph** tab displays visibility of each scene graph item as icons.

Connection Type	Connection Name	Function
Input	in	The object whose visibility setting you want to change.

Control (UI)	Default Value	Function
CEL	N/A	Specifies what part of the scene graph to assign this attribute to. The scene graph locations are specified using the Collection Expression Language (CEL). The **CEL** parameter options are available by clicking **Add Statements**. For more information, refer to the CEL Reference document found on the documentation HTML page (accessed through **Help** > **Documentation**) or the CEL Statement Widget Type in Common Parameter Widgets.
visible	1	Sets the visibility of objects in the render. 0 specifies not visible in render and anything else specifies visible in render.

End User License Agreement (EULA)

PLEASE READ THIS EULA CAREFULLY BEFORE ORDERING OR DOWNLOADING OR USING ANY SOFTWARE PRODUCTS OF FOUNDRY. YOUR ATTENTION IS PARTICULARLY DRAWN TO: (A) CLAUSE 8 IN WHICH SUBSCRIPTION CUSTOMERS AGREE TO THE AUTO-RENEWAL OF THEIR LICENSE ON AN ANNUAL BASIS; (B) CLAUSES 15 AND 16 WHERE WE LIMIT OUR LIABILITY TO USERS OF OUR SOFTWARE PRODUCTS; (C) CLAUSE 19.2 REGARDING THE DATA WE MAY COLLECT AND HOW WE MAY USE IT; AND (D) CLAUSE 19.3 WHERE YOU AUTHORISE FOUNDRY TO USE THE SOFTWARE TO ACCESS AND COLLECT CERTAIN INFORMATION FROM YOUR COMPUTER NETWORKS AND TO TRANSMIT THIS INFORMATION TO FOUNDRY.

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 PRODUCTS DIRECT FROM FOUNDRY: YOU HAVE THE RIGHT TO CANCEL YOUR CONTRACT AND OBTAIN A FULL REFUND IN ACCORDANCE WITH CLAUSE 9. HOWEVER YOU WILL LOSE THIS RIGHT ONCE YOU INSTALL THE SOFTWARE OR LOGIN TO AN INDIVIDUAL LOGIN OR TEAM LOGIN LICENSE. THIS DOES NOT AFFECT YOUR CONSUMER RIGHTS IN RELATION TO DEFECTIVE PRODUCTS OR SERVICES.

This END USER LICENSE AGREEMENT ("**EULA**") is, in cases where you purchase our product(s) direct from Foundry, incorporated into the agreement between The Foundry Visionmongers Ltd a company registered in England and Wales with company number 4642027 and whose registered office is at Squire Patton Boggs Secretarial Services Limited, Rutland House, 148 Edmund Street, Birmingham, United Kingdom, B3 2JR and whose address for correspondence is 5 Golden Square, London W1F 9HT, ("**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 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 Foundry and Licensee on which Licensee is permitted to use Foundry's product(s), including this EULA.

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 Foundry or to a reseller of Foundry, in connection with the Agreement, in connection with any other software license, in connection with any other agreed terms between Foundry and the Licensee, to use any Software product(s) of Foundry and/or in connection with any Maintenance and Support Agreement as defined in clause 11.1.

1. GRANT OF LICENSE

1.1 Subject to terms and the scope of the applicable licence model as set out in clause 2, the limitations of clause 3 and all the other terms of the Agreement, 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**") together with, and in accordance with, any accompanying user guide and other documentation (the "**Documentation**"), solely for Licensee's, where the Licensee is a business, own internal purposes or, where the Licensee is a consumer, domestic and private 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 by Foundry in writing. For the avoidance of doubt, 'use' in a Home Country requires the server, network and user to be based in the Home Country. 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: a) 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, and b) our Software may not be accessed or used in countries which are subject to import/export controls or sanctions, as further detailed in Clause 22.

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 or license entitlements 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 Software product that you purchase from Foundry, the product will be licensed (and not sold) to you on either a rental or subscription basis in accordance with one or more of the license models set out in this clause 2.1 and clause 2.2, as specified in Foundry's invoice or order confirmation (as applicable), and subject to the other terms and conditions of this EULA. Please note that some licensing models set out below do not apply to certain Software products of 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 by Licensee.

(a) "**Offline Node Locked License**"

If Licensee purchases an Offline Node Locked License, Licensee will install and use only a single copy of the Software on only one computer at a time in the Home Country.

(b) "**Modo Individual License**"

If Licensee has a Modo Individual License then: (a) Licensee warrants and represents that Licensee is a natural person and that only Licensee will use the Software; (b) Licensee may transfer or assign ("**transfer**") the Modo Individual License to another natural person ("**Assignee**") subject to Licensee: (i) notifying Foundry of such transfer and obtaining Foundry's express written consent, (ii) paying an administrative fee with respect to such transfer as may be required by 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; (c) Licensee shall not share its login details for the Software with any third party (d) Licensee shall be entitled to use the Software on different computers which may be located anywhere and use is not limited to the Home Country; (e) use of the Software shall be limited to no more than one concurrent use at all times.

(c) "**Offline Floating License**"

If Licensee purchases an Offline Floating License, then: (a) Licensee may use the Software on any number of computers, provided that the number of concurrent users shall never exceed the total number of valid Offline Floating Licenses purchased by Licensee; and (b) use of the Software shall be limited to any site in the Home Country.

(d) "**Individual Login License**"

If Licensee purchases an Individual Login 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). Use of the Software shall be limited to any site in the Home Country.

(e) "**Team Login License**"

If Licensee purchases a Team Login License, then: (a) Licensee may use the Software on any number of computers, provided that the number of concurrent users shall never exceed the total number of valid Team Login Licenses purchased by Licensee; (b) use of the Software shall be limited to any site in the Home Country; and (c) use of the Software, and changes to the user authorizations within Licensee's team organization(s), shall be in accordance with any Team Login License guidance and instructions, as published on Foundry's website and which may be amended from time to time.

(f) "**Nuke Indie License**"

If Licensee purchases a License for Nuke Indie, then: (a) Licensee warrants and represents that Licensee (i) is a natural person and that only Licensee will use the Software; (ii) is working independently and shall not use the Nuke Indie License in a pipeline with other Nuke commercial or Nuke Indie licenses, whether those licenses are held by the Licensee, other individuals or other businesses or organisations; and (iii) earns less

than $100,000 USD (or local equivalent) a year; and (iv) that Licensee satisfies all criteria set out in Foundry's Nuke Indie Eligibility Requirements as published on its website and which may be amended from time to time (the "**Nuke Indie Eligibility Requirements**"); (b) Licensee shall not share its login details for the Software with any third party; (c) Licensee shall not purchase or use more than one Nuke Indie License; (d) Licensee may use the License on different computers, subject to (i) a maximum of two computer authorisations at any one time and (ii) no more than one concurrent user at any one time; and (e) Licensee shall use the Software in accordance with terms of the Nuke Indie Eligibility Requirements, including abiding by any functional restrictions; and (f) the provisions of clause 8 shall apply.

(g) "**Modo Subscription License**"

If Licensee has a Modo Subscription License, then: (a) Licensee warrants and represents that Licensee is a natural person and that only Licensee will use the Software; (b) Licensee shall not share its login details for the Software with any third party; (c) Licensee may use the Software on different computers which may be located anywhere and use is not restricted to the Home Country; (d) Licensee may use the License on different computers, subject to (i) a maximum of two computer authorisations at any one time and (ii) no more than one concurrent user at any one time; (e) Licensee shall not purchase or use more than one Modo Subscription License; and (f) the provisions of clause 8 shall apply.

(h) "**Mari Individual Subscription License**"

If Licensee purchases a Mari Individual Subscription License then: (a) Licensee warrants and represents that Licensee is a natural person and that only Licensee will use the Software; (b) Licensee shall not share its login details for the Software with any third party; (c) Licensee may use the Software on different computers which may be located anywhere and use is not restricted to the Home Country; (d) Licensee may use the License on different computers, subject to (i) a maximum of two computer authorisations at any one time and (ii) no more than one concurrent user at any one time; (e) Licensee shall not purchase or use more than one Mari Individual Subscription License; and (f) the provisions of clause 8 shall apply.

(i) "**Rental License**"

If Licensee has purchased the Software on a rental basis, the License shall be limited to the term of the rental as agreed in writing with Foundry after which it shall automatically expire.

(j) "**Educational License**"

If Licensee has purchased the Software on the discounted terms of Foundry's educational program and eligibility criteria as published on its website and amended from time to time, Licensee warrants and represents to Foundry as a condition of the Educational License that: (i) it is an educational institution that will use the Software only for the purpose of training and instruction, and for no other purpose, (ii) Licensee will at all times comply with any eligibility criteria and guidelines in respect of the Educational License, as may be communicated to a Licensee and/or published on Foundry's website and amended from time to time, and (iii) use of the Software shall be limited to any site in the Home Country. Unless the Educational License is a Floating License, Licensee shall use the Software on only one computer at a time.

(k) "**Graduate License**"

If Licensee has purchased the Software on the discounted terms of Foundry's Graduate Licence, Licensee warrants and represents to Foundry as a condition of the Graduate Licence that the Licensee has graduated from a university or tertiary education institute no more than six (6) months prior to the grant of the Graduate Licence. The Licensee may use the Software in a personal capacity only (namely, as a sole trader or freelancer). For the avoidance of doubt, Licensee may not use the Graduate License in connection with any work or employment, whether paid or unpaid, or other such collaborations with studios or similar organisations. Foundry reserves the right to require evidence of graduation by the Licensee. Use of the Software shall be limited to any site in the Home Country.

(l) "**Student License**"

If Licensee has purchased the Software on the discounted terms of Foundry's student program and eligibility criteria as published on its website and amended from time to time, Licensee warrants and represents to Foundry as a condition of the Student License that: (i) the Licensee 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) Licensee will at all times comply with any eligibility criteria and guidelines in respect of the Student License, as may be communicated to a Licensee and/or published on Foundry's website and amended from time to time, and (iii) use of the Software shall be limited to any site in the Home Country. Unless the Student License is a Floating License, Licensee shall use the Software on only one computer at a time.

(m) "**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.

(n) "**Mari indie**"

Variants of Mari with limited functionality as described in the Documentation are available to purchase. If Licensee has purchased such a variant, Licensee warrants and represents to 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 the Software; and (iv) only Licensee will use the Software.

(o) "**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 is for product evaluation and learning purposes only and may only be used by a natural person, A Trial License lasts a limited specified period on the expiry of which the Software will automatically cease to function. Foundry may terminate any Trial License for convenience immediately on notice to Licensee. Licensee will use the Software for product evaluation and learning purposes only.

(p) "**Subscription License**"

If Licensee has purchased the Software on a subscription basis then: (a) the License shall be limited to the Subscription Period (as defined in clause 8.1) after which it shall automatically expire and; (b) the provisions of clause 8, including but not limited to auto-renewal, shall apply.

(q) "**Nuke Stage License**"

If Licensee purchases a Nuke Stage License, then: (a) Licensee may use the Software on any number of computers, provided that the number of concurrent Nuke Stage Output Device(s) shall never exceed the total number of valid Nuke Stage Floating Licenses purchased by Licensee; and (b) use of the Software shall be limited to any site in the Home Country. "Nuke Stage Output Device(s)" shall mean one region designated for rendered output, which may be mapped to one endpoint such as display windows, SDI outputs, files, or other render targets.

2.2 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, Headless Rendering licenses may be used on any number of computers, provided that the number of concurrent computers shall never exceed the total number of valid Headless Rendering licenses purchased by Licensee.

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. Licensee is authorised 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 in each case except as and to the extent that applicable law requires such reverse engineering, decompilation, disassembly or discovery to be permitted and it is not lawful to contract out of such requirement; (c) implement or use any method or mechanism designed to enable product functionality not available in the Software but available in (i) other Foundry products; or (ii) other Foundry releases of the

same product; (d) (subject to clause 4) modify, adapt, translate or create derivative works based on the Software or Documentation; (e) 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; (f) 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; (g) copy or install the Software or Documentation other than as expressly provided for in this EULA; or (h) 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 Foundry or any third party with intellectual property rights in the Software (each, a "**Third Party Licensor**"); (i) use the Software, or permit any third party to use the Software, for any illegal purpose. The Licensee shall maintain appropriate internal security, safeguards and controls which shall be at least as great as the measures the Licensee uses to keep Licensee's own information secure (but in any case, using no less than a reasonable degree of care) to ensure that the Software is not used in any way which is in violation of this clause 3 or of this Agreement. For the avoidance of doubt, Licensee remains fully liable for any unauthorized use of the Software by its staff. For purposes of this clause 3, the term "Software" shall include any derivatives of the Software.

Notwithstanding clause 3(b) above, where the reduction of the Software to human readable form is necessary for the purposes of integrating the operation of the Software with the operation of other software or systems used by the Licensee in accordance with section 50B of the Copyright Designs and Patents Act 1988 (or any analogous legislation in other jurisdictions), prior to reducing the Software to human readable form (whether by reverse engineering, decompilation or disassembly), the Licensee shall notify Foundry in advance and allow Foundry a reasonable period to either carry out such action as is required to fulfil the integration or provide the information necessary to achieve such integration to the Licensee, and in either case the Licensee shall meet Foundry's reasonable costs in doing so.

Unless Licensee has purchased an Individual License, a Team Login License or an Individual Login License, for other license types, if the Software is moved from one computer to another (or, in the case of an Offline Floating License, from one license server to another), the issuing of replacement or substituted license keys or approval of the new server location, is in Foundry's sole discretion and is subject to and strictly in accordance with Foundry's License Transfer Policy, which is available on Foundry's website and which requires a fee to be paid in certain circumstances. Foundry may from time to time and at its sole discretion vary the terms and conditions of the License Transfer Policy.

The Software and/or any components thereof are not intended to be used for and Licensee shall not use the Software for (or modify the Software such that is suitable for): (i) prohibited practices under the European Union Artificial Intelligence Act (the "**EU AI Act**") or similar legislation, (ii) any use that would result in the Software being considered a high-risk AI system or that otherwise qualifies as "high-risk" under the EU AI Act. Licensee may not remove, alter or obscure any metadata fields which are automatically applied to content generated by the Software which indicate that such content has been artificially generated or manipulated.

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 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 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, where the Licensee is a business, its own internal business purposes or, where the Licensee is a consumer, domestic and private 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 Foundry at any time during the Agreement (whether imposed or requested by Foundry or by any Third Party Licensor);

4.2 to the extent that the Software links to or itself incorporates any open source software and/or software libraries ("**OSS Components**") that are provided to Licensee with or as part of the Software, then where such OSS Components are licensed on the terms of an open source software licence that requires Foundry to make the OSS Components available to the Licensee on specific terms (the "**OSS Licence Terms**"), those OSS Components are licensed to Licensee on, and subject to, the terms of the relevant OSS Licence Terms;

4.3 where Foundry is required by any OSS Licence Terms to make the source code of the relevant OSS Component available to the Licensee, 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 Foundry a charge that does not exceed Foundry's costs of doing so, provide Licensee with the source code of the relevant OSS Component (the "**OSS Source Code**") in order that Licensee may modify the OSS Component in accordance with the relevant OSS Licence Terms, together (where appropriate) with certain object code of the Software necessary to enable Licensee to re-link any modified OSS Components to the Software (the "**Object**"); and

4.4 notwithstanding any other term of the Agreement, Foundry gives no express or implied warranty, undertaking or indemnity whatsoever in respect of the Source Code, the OSS Components, the OSS Source Code 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 Components or the OSS Source Code 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, Foundry shall have no obligation to provide support, maintenance, upgrades or updates of or in respect of any of the Source Code, the OSS Components (save for any obligations Foundry may have in respect of any elements that form part of the Software as a whole), the OSS Source Code, the Object or any Modification. Licensee shall indemnify Foundry against all liabilities and expenses (including reasonable legal costs) incurred by 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 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 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 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 Foundry or any Third Party Licensor, and Licensee shall be strictly prohibited from using the name, trademarks or service marks of Foundry or any Third Party Licensor in Licensee's promotion or publicity without Foundry's prior express written approval.

Subject to clause 4.3, Foundry undertakes (the "**Undertaking**") to defend Licensee or at 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 Foundry. The Undertaking is conditional on Licensee giving written notice of the Claim to 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 Foundry's reasonable opinion is likely to be made, against Licensee, 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 by Licensee. The Undertaking constitutes Licensee's exclusive remedy and Foundry's only liability in respect of any Claim.

7. LICENSE FEE

7.1 Licensee acknowledges that the rights granted to Licensee under this EULA are conditional on Licensee's timely payment of the license fee payable to Foundry in connection with the Agreement or, as the case may be, payable to Foundry's reseller (the "**License Fee**"). Except where the applicable invoice expressly sets out payment in instalments, the License Fee shall be payable in full as one single payment.

7.2 Licensee will be charged and agrees to pay to Foundry or Foundry's authorized reseller (as applicable): (a) the License Fee as notified by Foundry (or its reseller) at the time of the initial purchase of the License; and (b) in respect of any Subscription Auto-renewal Period for a Subscription License, the License Fee as notified by Foundry (or its reseller) on or about the applicable Renewal Date.

7.3 Unless stated otherwise, any Licence Fee notified to the Licensee by Foundry (or its reseller) is exclusive of sales tax, VAT and any other similar taxes, duties or levies, which shall be payable by the Licensee (and the Licensee agrees to pay) in addition to the Licence Fee.

7.4 In the cases of Non-Commercial Licenses or Trial Licenses or specific End of Life Software, the fact that no License Fee may be payable shall not be construed as a waiver by Foundry of any right or remedy available to it in relation to any breach by the 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. SUBSCRIPTION LICENSES AND AUTO-RENEWAL

8.1 If Licensee has purchased a Subscription Licence, the License shall be limited to the Initial Subscription Period and any/all Auto-renewal Periods (each as defined below) (together the "**Subscription Period**") after which it shall automatically expire.

8.2 The Subscription Licence shall begin as soon as Foundry accepts Licensee's order by issuing Licensee with a license key (the "**Subscription Start Date**") and shall continue for an initial period of twelve (12) months (the "**Initial Subscription Period**") unless earlier terminated in accordance the terms of this EULA, or unless otherwise communicated by Foundry.

8.3 Unless Licensee opts out of auto-renewal in accordance with clause 8.5 then upon the first anniversary of the Subscription Start Date and each subsequent anniversary (each a "**Subscription Renewal Date**"), Licensee's Subscription Licence shall renew automatically for a further twelve (12) months (each an "**Auto-renewal Period**"). Licensee's Subscription License will continue to auto-renew in this manner until Licensee opts out of auto-renewal or unless earlier terminated in accordance with the terms of this EULA.

8.4 Prior to each Subscription Renewal Date, Foundry shall send one (1) email to advise you that your Subscription License is approaching auto-renewal to the contact email address as provided by Licensee in accordance with clause 23. The reminder email will be sent not less than thirty (30) days prior to the relevant Subscription Renewal Date.

8.5 **Opting Out of Auto-renewal and License Expiry.** If Licensee wishes to opt out of auto-renewal then you must email licenses@foundry.com providing details of the Subscription Licence(s) which you wish to opt out not less than seventy-two (72) hours prior to the relevant Subscription Renewal Date. Provided that Licensee notifies Foundry in accordance with the provisions of this clause 8.5 then your Subscription License (s) will not auto-renew and shall expire at the end of the then-current Subscription Period.

8.6 **Increases to the License Fee for Subscription Licenses.** Foundry reserves the right to increase the License Fee for Subscription Licenses from time to time provided that it shall provide Licensee with not less than thirty (30) days' notice of any increase prior to the relevant Subscription Renewal Date.

9. CANCELLATIONS

9.1 Licensee may cancel a License within 14 days of the original purchase date to obtain a full refund and Licensee will no longer be able to use the Software from the cancellation date. Licensee's right to obtain a refund will be lost once the Software has been installed.

9.2 Refunds are not payable for cancellations made after such date. This includes Subscription licenses which are subject to the fixed twelve (12) month terms and for which the Licensee may opt out of auto-renewal in accordance with Clause 8.

9.3 Cancellations and requests for refunds can be made by contacting Foundry's Sales Support team at licenses@foundry.com.

10. END OF LIFE

10.1 From time to time, Foundry will retire Software product(s) (and any/ all versions) by withdrawing them from the market. Such Software product(s) will be designated as end of life ("**EoL**") and a corresponding announcement (the "**Announcement**") will be published on Foundry's website from time to time (the date of such announcement being the "**EoL Notification Date**"). The announcement will include an EoL date ("**EoL Date**") and other relevant information.

10.2 No new License(s) (including renewals, if applicable) for the EoL Software product(s) will be sold after the EoL Notification Date. Unless otherwise provided for in the Announcement, the EoL Date will be the last day that the EoL Software product(s) is sold, improved, maintained, updated, or supported (subject to the below).

10.3 Unless otherwise provided for in the Announcement, a Licensee with an active Subscription License(s) at the time of the EoL Date will receive certain limited support until the relevant Subscription Period expires. A Licensee with an active perpetual License(s) (purchased on or prior to 31 December 2023) for which it has renewed (or auto renewed) the associated Maintenance and Support Agreement prior to the EoL Notification Date, will receive certain limited support until the EoL Date, or for a period specified in the Announcement.

10.4 In the event of EoL, Foundry advises Licensee to install the latest product releases, security updates, fixes and patches available until the EoL Date, to remain as secure as possible. Older products may not meet security or performance requirements, and Foundry cannot confirm compatibility with future operating system updates after the EoL Date. To the maximum extent permitted by applicable law, Foundry shall not be liable for any form of loss, damage or disruption in respect of the EoL Software product(s) beyond the EoL Date.

10.5 Foundry retains all rights, including intellectual property rights, associated with the EoL Software product(s) beyond the EoL Date. Any use by Licensee of the EoL Software product(s) beyond the EoL Date will continue to be governed by the terms of this Agreement.

11. MAINTENANCE AND SUPPORT

11.1 Subject to Licensee's timely payment of the applicable License Fee, each Subscription Licence and each Rental License shall include access to certain maintenance and support services in accordance with the Maintenance and Support Agreement which is available on Foundry's website (the "**Maintenance and Support Agreement**").

11.2 If Licensee purchased a perpetual License on or prior to 31 December 2023 for which it has renewed (or auto-renewed) the associated Maintenance and Support Agreement, then such License shall include access to certain maintenance and support services for the relevant continuous Renewal Support Period(s) (as defined in, and in accordance with, the Maintenance and Support Agreement), after which Licensee shall not be entitled to purchase further maintenance and support for such License.

11.3 Foundry may vary the terms and conditions of the Maintenance and Support Agreement from time to time and at its sole discretion.

12. TAXES AND DUTIES

Licensee agrees to pay, and indemnify 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.

13. LIMITED WARRANTY

13.1 Subject to clause 13.3, Foundry warrants that, for a period of ninety (90) days after Licensee first downloads the Software ("**Warranty Period**"): (a) 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 describes the operation of the Software in all material respects. If, within the Warranty Period, Licensee notifies 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, 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, and Foundry shall have no liability for, any defect or fault caused by: (a) unauthorised use of or any amendment made to the Software by any person other than Foundry; and/or (b) use of the Software in conjunction with third party technology. 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.

13.2 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.

13.3 If Licensee purchases a license of the Software that is of a fixed term duration, the Warranty Period in clause 13.1 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.

14. INDEMNIFICATION

14.1 Licensee agrees to indemnify, hold harmless and defend Foundry, the Third Party Licensors and 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 use including but not limited to modification of the Software or Documentation by Licensee or any person connected with Licensee or any person provided with the Software by Licensee, infringes the intellectual property rights or other proprietary rights of any third party.

14.2 Licensee agrees to indemnify, hold harmless and defend Foundry, the Third Party Licensors and 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 use (including but not limited to modification) of the Software or Documentation, by Licensee, any person connected with Licensee or any person provided with the Software by Licensee, causes any of them or any third party (including but not limited to those viewing a product of the Software) to suffer harm or loss of any kind.

15. LIMITATION OF LIABILITY TO BUSINESS USERS

This clause applies where Licensee is a business user (and for these purposes any Licensee that is not a consumer shall be treated as 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 Licensee 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 the Software (whether directly or indirectly) to third parties. Foundry shall not under any circumstances whatever be liable to Licensee, its affiliates, officers, directors, shareholders, employees, agents or other representatives, 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 (in each case whether the loss is direct or indirect) or any indirect or consequential loss or damage. In respect of any other losses, 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$5,000 (five thousand USD) and a sum equal to the License Fee. Nothing in the Agreement shall limit or exclude Foundry's liability for death or personal injury resulting from our negligence, for 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 Documentation. Except as expressly stated in writing in this EULA, there are no conditions, warranties, representations or other terms, express or implied, that are binding on 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.

16. 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 Licensee 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 Foundry has no liability to Licensee for any loss of profit, loss of business, business interruption, loss of business opportunity or loss or harm caused to property or data used for professional purposes. Foundry is only responsible for loss or damage suffered by Licensee that is a foreseeable result of Foundry's breach of the Agreement or its negligence but Foundry is not responsible for any loss or damage that is not foreseeable. 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$5,000 (five thousand USD) and a sum equal to the latest annual License Fee.

Nothing in the Agreement shall limit or exclude Foundry's liability for death or personal injury resulting from our negligence, for fraud or fraudulent misrepresentation or for any other liability that cannot be excluded or limited by applicable law.

17. TERM; TERMINATION

17.1 The Agreement is effective upon Licensee's download of the Software, and the Agreement will remain in effect until termination or expiry. Licensee may terminate the Agreement on written notice to Foundry if Foundry is in material breach of this Agreement and fails to cure the breach within 10 (ten) working days of receiving notice of such breach. If Licensee breaches the Agreement, Foundry may terminate the License immediately by notice to Licensee.

17.2 If the Agreement expires or is terminated, the License will cease immediately and Licensee will immediately cease use of any Software and Documentation and either return to Foundry all copies of the Software and Documentation in Licensee's possession, custody or power or, if Foundry directs in writing, destroy all such copies. In the latter case, if requested by Foundry, Licensee shall provide Foundry with a certificate confirming that such destruction has been completed within 10 days.

17.3 Foundry reserves the right to terminate and/or suspend the License as it deems reasonable in its sole discretion by notice to Licensee if it becomes aware that Licensee has failed to pay any sum due either to Foundry or to a reseller of Foundry either in connection with the Agreement or in connection with any other Software license to use any product(s) of Foundry, in connection with any other agreed terms between Foundry and the Licensee in connection with the Software, in connection with any Maintenance and Support Agreement or if the Licensee is otherwise in breach of or fails to comply with any term of the Agreement.

17.4 Foundry may also terminate this Agreement if Licensee becomes subject to bankruptcy proceedings, becomes insolvent, or makes an arrangement with Licensee's creditors. This Agreement, and for the avoidance of doubt the Licenses to which it relates, will terminate automatically without further notice or action by Foundry if Licensee goes into liquidation.

18. 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 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 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 for the purposes of that Authorized Project, 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 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. 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 (other than as a result of the Licensee's breach of its obligations under the Agreement), 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 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.

19. INSPECTION AND INFORMATION

19.1 Unless Licensee is a consumer, Licensee shall advise Foundry on demand of all locations where the Software or Documentation is used or stored. Licensee shall permit Foundry or its authorized agents to audit all such locations during normal business hours and on reasonable advance notice.

19.2 The Software may include mechanisms to access and collect limited information from computer(s) on which it is installed and from any IT systems to which those computer(s) may be connected (including any system registry files) and transmit it to Foundry and/or its resellers, including the ability to locally cache such information on such computers. Such information (the "**Information**") may include details of relevant license (s) to Foundry products, details of computer and network equipment, details of the operating system(s) in use on such computer equipment, email domain relating to owners of such computer and network equipment, the location of the computer(s) on which the Software is installed and the profile and extent of use of the different elements of the Software and other Foundry software. 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, (d) to confirm the identity of Licensee, to identify unlicensed use of the Software (including use of pirated or other unlicensed copies of the Software) and to assist Foundry (and its resellers and any enforcement bodies) in contacting any unlicensed users of the Software and seeking to terminate unlicensed use of the Software, and (e) to advise Licensee about service issues such as available upgrades and maintenance expiry dates. To the extent that any Information constitutes personal data for the purposes of the General Data Protection Regulation (EU) 2016/679 ("**EU GDPR**") and the version of the EU GDPR retained in UK domestic law as further defined in the Data Protection Act 2018 ("**UK GDPR**"), in each case, as amended, superseded or replaced from time to time (as applicable, "**GDPR**") it shall be processed in accordance with the GDPR and with Foundry's Privacy Notice

(see https://www.foundry.com/privacy-notice), as may be updated by Foundry from time to time. Licensee undertakes to make all of users of the Software aware of the uses which Foundry will make of the Information and of the terms of Foundry's Privacy Policy.

19.3 By downloading or using the Software, you (i) warrant that you are entitled to control access to the computer(s) on which the Software is downloaded and any IT systems to which they may be connected, and (ii) irrevocably authorise Foundry (through the use of the Software) to access such computer(s) and IT systems (including any system registry files) and collect the Information from them and to transmit that Information to Foundry and its resellers (and any enforcement bodies) and use it for the purposes identified at clause 19.2(a) to (e) above.

20. 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 - 2025 The Foundry Visionmongers Ltd. All Rights Reserved.

* Unpublished- rights reserved under the Copyright Laws of the United Kingdom.

21. SURVIVAL

Clause 6, clause 7 and clauses 12 to 25 inclusive shall survive any termination or expiration of the Agreement.

22. IMPORT/EXPORT CONTROLS AND SANCTIONS

To the extent that any Software made available under the Agreement is subject to restrictions upon export and/or re-export from any applicable jurisdiction (including the United States), Licensee agrees to comply with, and not act or fail to act in any way that would violate, applicable international, national, state, regional or local laws and regulations, including, without limitation, the U.S. Export Administration Act and the Export Administration Regulations, the regulations of the U.S. Department of Treasury Office of Foreign Assets Control, the International Traffic in Arms regulations, the United States Foreign Corrupt Practices Act, the UK Export Control Act 2002 and the UK Export Control Order 2008 (collectively, "**Export Laws**") as those laws

may be amended or otherwise modified from time to time, and neither 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. Without limiting the foregoing, Licensee agrees that it will not sell, supply, transfer, export or re-export, directly or indirectly, Foundry's Software or related products and services, or any commodity, technology, technical data, software or service that incorporates, contains or is a direct product of Foundry's Software, products and/or services, (i) in violation of the Export Laws; (ii) to any country for which an export license or other governmental approval is required at the time of export, without first obtaining all necessary export licenses or other approvals; (iii) to any country, or national or resident of a country, to which trade is embargoed by the United States and/or sanctioned by the United Kingdom or the European Union; (iv) to any person or firm on any government agency's list of blocked, denied or barred persons or entities, including but not limited to the U.S. Department of Commerce's Denied Persons List and Entities List, and the U.S Treasury Department's Specially Designated Nationals List; or (v) to the Russian Federation and/or Belarus or for use in the Russian Federation and/or Belarus, or (vi) for use in any nuclear, chemical or biological weapons, or missile technology end-use. Licensee warrants and represents that it is not subject to any Export Law, sanction and/or other form of trading compliance restriction or regulation which may limit or prohibit its right to download, install and/or use the Software in accordance with this EULA.

23. 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, by Foundry, at any time.

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 Foundry fails to insist that Licensee performs any obligation under the Agreement, or delays in doing so, that will not mean that Foundry has waived its rights.

If any provision or part-provision of this Agreement is or becomes invalid, illegal or unenforceable, it shall be deemed deleted, but that shall not affect the validity and enforceability of the rest of this Agreement.

Unless Licensee is a consumer, Licensee agrees that 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. Foundry agrees to promptly cease any such use on Licensee's written request.

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 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. 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.

Email Address for General Correspondence: Licensee shall notify Foundry of an email address for the provision of any correspondence in connection with the Agreement and shall notify Foundry via licenses@foundry.com of any change(s) to that email address. Please note, the email address you provide is important for the provision of information and notices to you, including in relation to the autorenewal of any Subscription License (if applicable). It is your responsibility to provide and maintain an up-to-date email address. Foundry shall store details of and may use the email address to notify you in accordance with the terms of this Agreement.

24. NOTICES

Any legal notices from the Licensee to Foundry should be sent via postal mail or international delivery to The Foundry Visionmongers Ltd, 5 Golden Square, London, W1F 9HT, addressed to The General Counsel. These notices will be effective upon receipt by Foundry. Additionally, a copy of the notice should be emailed to Foundry at legal@foundry.com. Unless otherwise agreed in writing, any notices from Foundry to the Licensee will be provided either (a) by email to the registered email address associated with the Licensee's account, or (b) in any other reasonable manner that involves specific notification to the Licensee. Notices from Foundry to the Licensee will be effective (i) one day after being sent by email, and (ii) five days after being posted or sent by international delivery service. The Licensee agrees that service of process may be effected on the Licensee by registered mail sent to the last known address on record with Foundry, if permitted by applicable law.

25. COMPLAINTS & ONLINE DISPUTE RESOLUTION PLATFORM

We hope that you are satisfied with any Software purchase made or service received from Foundry, but if you have a complaint, in the first instance, please contact us on licenses@foundry.com or through our Support Portal: https://support.foundry.com/hc/en-us (for technical support and bug reports), or you can request a call back from the Sales team here: https://www.foundry.com/contact-us.

Last updated 27 March 2025.

Copyright © 27 March 2025 The Foundry Visionmongers Ltd.

All Rights Reserved. Do not duplicate.