
# WHAT'S NEW IN KATANA

VERSION 2.0V1

THE FOUNDRY.

The Foundry

5 Golden Square,

London,

W1F 7HT

Rev: 13 May 2015

# CONTENTS

# Introduction

This document covers a technical overview of new features and important changes since Katana 1.x.

Katana 2.0 introduces many changes, both at the core and at the UI level. These changes focus on several areas:

- **Interactivity and performance** - threading and data re-use
- **More flexible and open APIs for Katana plug-ins** - the Op API, new widget types
- **Workflow** - improvements to multi-pass setup, lighting, and general UI tweaks

At the heart of these changes is a brand new implementation of the scene graph processing engine Katana uses to build scene data: Geolib3.

NOTE: As the new implementation of the scene graph processing engine in Katana 2.0 constitute a substantial change, please make sure to thoroughly read the Katana Release Notes and supporting documentation, as your scenes and plug-ins may require modification.

## Geolib3: Overview

Geolib3 draws on the breadth of experience from building and using Geolib and Geolib2 in production. It is designed to better take advantage of the resources now available with modern hardware.

The most noteworthy Geolib changes are the persistence and re-use of data while working with Katana's node graph, and the ability to strategically adapt behavior to make best use of CPU resource and memory depending on the situation. For example, we use one approach when working in the UI to ensure the interface stays responsive and another during batch renders so they remain as efficient as possible.

NOTE: For more details, see the Geolib3: Into the Details section.

NOTE: Importantly, Geolib3 also expands the kinds of custom nodes you can develop as a facility. See The Op API section for more information.

Katana 2.0 leverages the increased functionality and performance offered by Geolib3 to provide you with a range of improvements, optimizations, and new features. The following sections give a brief overview of these changes.

## Interactivity and Performance

A key change for an artist using the Katana UI is that scene graph processing is now carried out asynchronously on another thread. This means far fewer blocks or stalls in the UI while working on a project. Additionally, we make much better use of previously computed data and therefore the parameter updates can be much more responsive. In many cases the **Viewer** tab and other tabs now are updated during drag, rather than on pen-up, providing quicker and more accurate feedback.

In addition to increased interactivity, Geolib3's architecture has also been designed to take advantage of modern systems with multiple CPU cores. The existing `FnScenegraphIterator` interface, already used throughout renderer plug-ins, has been made thread-safe and capable of being called efficiently and concurrently across multiple threads. Work is ongoing to add further performance gains from both the underlying Geolib3 runtime, individual Ops, and the renderer plug-ins themselves.

# New Workflow Features

## Graph State Variables

Node types and UI elements have been added to support the setting and querying of new, node graph-level Graph State Variables. These can be used to control which nodes in the node graph contribute to scene graph processing at a particular time. This can greatly simplify multi-pass/layer workflows or any other tasks that might want to re-use node setups with different inputs or outputs.

Graph State Variables can be set globally at the project level or locally at specific nodes in the node graph. Setting values locally changes the value at that node and any node upstream.

### New Node Types Related to Graph State Variables

- **VariableSet** - Sets values for Graph State Variables locally at this point in the node graph. This affects the value at this node and any node directly upstream.
- **VariableSwitch** - Switches which input is active depending on the value of a Graph State Variable.
- **VariableEnabledGroup** - Allows you to enable or bypass Group nodes depending on the value of a Graph State Variable.

### New UI Elements to Manage Project-wide (global) Graph State Variables

The main menu bar provides a widget showing the currently active project-wide Graph State Variables and their values. The variables shown here are the global variables set up in the **Project Settings** tab, not the local Graph State Variables set using, for example, VariableSet nodes in the node graph.

A **variables** parameter has been added to the **Project Settings** tab for you to create, edit, and delete global Graph State Variables.

### New UI Elements to Display Graph State Variables Based on the Currently Viewed Node

A Graph State Variables button has been added to the header of parameters of nodes edited in the **Parameters** tab. Clicking the button opens a pop-up widget that shows Graph State variables that are set up in **Project Settings** (globals), as well as those that are active in the node graph branch to which the respective edited node belongs, taking into account the currently viewed node (locals).

A menu item named **Dim Nodes Not Contributing to Viewed Node** has been added to the **Edit** menu of the **Node Graph** tab. This menu can be used to dim nodes that are not contributing to the currently viewed node, taking into account the states of Switch and VariableSwitch nodes.

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.

## GafferThree

A new Gaffer node type, named GafferThree, has been created to provide improved performance when dealing with large numbers of lights in Katana projects. The GafferThree implementation takes full advantage of the new scene graph processing library Geolib3.

NOTE: The existing legacy Gaffer node type from Katana 1.x is still present, and previously created projects should continue to work, but it is advised to move to using the new GafferThree node type where possible.

As well as performance, significant features of the GafferThree node type include the following:

- Lights, rigs, and master materials can be created and managed underneath an arbitrary root location in the scene graph. In the previous Gaffer node type, all lights would be created under **/root/world/lgt**. The new feature allows the creation of lights under separate branches in the scene graph.
- Adoption of lights from the incoming scene. This allows a GafferThree node to edit the material, geometry, linking, and transformation parameters, and also mute and solo state of lights, rigs, and master materials created in upstream GafferThree nodes.
- Ability to add child lights, rigs, and master materials under adopted rigs.
- Soloing a light in a GafferThree node affects downstream lights created by other GafferThree nodes.
- Parameter values of multiple selected items in the GafferThree object table can now be changed at once by changing the parameter value for one of the selected items.
- Color swatches in the **Color** column of the GafferThree object table in the **Parameters** tab are shown with filmlook visualization/display transform turned on.
- The icon for a light in the GafferThree indicates its mute state, which is consistent with the **Scene Graph** tab.
- The **Linking** tab provides fine-grained control over light and shadow linking. The tab presents two CEL widgets, **on** and **off**, to specify locations for light linking, and a **clearUnmatched** checkbox that removes any locally set light linking information on locations that are not matched by either of the CEL expressions. If a location is matched by both the on and the off CEL expressions, then on overrides off.
- The **Link** column in the GafferThree object table provides a read-only indication of the light and shadow linking settings for each light.
- Callbacks can be registered for the following actions:

- `onGafferLightCreated` - executed when a light is created.
- `onGafferRigCreated` - executed when a rig is created.
- `onGafferMasterMaterialCreated` - executed when a master material is created.
- `onGafferShaderSelected` - executed when a shader is selected in the GafferThree object table.

- Colors are used for items in the GafferThree object table to indicate where a value has come from:
  - **Gray/white** - default value.
  - **Yellow** - locally set value.
  - **Blue** - forced default value.
  - **Pink** - value inherited from a referenced master material.
- Right-clicking a cell in the GafferThree object table shows a context menu with useful commands for manipulating underlying parameters. The context menu in the **Shader** column adds a sub-menu for assigning a shader. Custom columns can define their own context menu through the `createContextMenu()` method on their item delegate.

## 2D and 3D Update Modes

Katana's **2D Render Mode** and **3D Live Render Mode** features have been replaced with **2D Render Mode**, and the application-wide **3D Update Mode** respectively. These modes govern the submission of parameter edits for processing by the 2D rendering system and Geolib3.

- In **Manual** mode, pending parameter edits are indicated by the **Trigger 2D Update** and **Trigger 3D Update** buttons, and are committed by pressing these buttons.
- The **3D Update Mode** button is displayed in the main menu bar, as well as in the **Scene Graph** and **Monitor** tabs, and applies to all edits of parameters of 3D nodes.

The **3D Update Mode** replaces the **Disable Scenegraph Updates** button previously available in the **Scene Graph** and **Viewer** tabs.

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 correctly 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.

## Python Tab

The interactive **Python** tab has been enhanced to support:

- In-line code completion,
- Indentation of blocks of code with the **Tab** key,

- Un-indentation of blocks of code with **Shift**+**Tab,**
- More comprehensive syntax highlighting, and
- Tooltips which display introspection data when the pointer is over the tab.

Preferences for the **Python** tab in the **python** category of preferences have been revised:

- The **commandFont** and **resultFont** preferences have been removed, as it was not possible to change them from the **Preferences** dialog. The appearance of text in the **Python** tab now depends on the application font preference.
- Preferences have been added to control the auto-completion behavior (**autoCompletionBehavior**) as well as whether or not to show help tooltips in the **Python** tab (**showHelpTooltips**).

# Lua OpScripts

Due to the fact that the CPython GIL has performance implications when multi-threaded, Katana 2.0 introduces a Lua-based OpScript node type that can be used instead of Python-based AttributeScript nodes.

It is recommended to use OpScript nodes wherever possible as they are more performant and also have access to the full Op API, allowing creation of scene graph locations, reading data from multiple inputs, and other powerful features.

# Live Rendering

## Live Render Controls

The **Live Render Control** tab of previous Katana releases has been replaced with several more Katana-native, more flexible features. These include:

- A new column in the **Scene Graph** tab that allows you to toggle locations to generate Live Rendering updates when their attributes change. This allows much finer control over how many updates are emitted to the renderer plug-ins and which locations they come from.
- New GenericAssign-based node types named **<renderer>LiveRenderSettings** that are used to set settings for Live Rendering. These node types replace the renderer-specific parameters that were previously displayed in the **Live Render Control** tab.
- The custom Live Rendering command buttons that were previously implemented in the renderer info plug-ins and displayed in the **Live Render Control** tab have now been replaced with Qt menu actions which derive from the new **BaseLiveRenderAction** Python class and are placed in the **Live Render** menu in the **Monitor** tab and also in the **Live Rendering** sub-menu of the **Render** main menu. This allows much greater flexibility in the implementation of custom Live Render actions than before, particularly with access to the new **LiveRenderAPI** functions.
- The option to use the same camera as the **Viewer** tab has been replaced by a button at the bottom of the **Viewer** tab, named **Live Render from Viewer Camera**.

## Modifications of Interactive Render Filters (IRFs) during Live Rendering

In previous releases of Katana, Interactive Render Filters (IRFs) could not be modified during a Live Render session. For example, changes in the order of IRFs in the IRFs pop-up widget and changes to parameters of RenderFilter nodes were not reflected in the Live Render.

IRFs have now been revised to be based on terminal Ops that are added to the Geolib3 client used in Live Rendering, and are now taken into account during a Live Render session. This means that it is possible to change the order of RenderFilters or to change any of their parameters, and have those changes be reflected in the rendered image.

# New APIs and Widgets

Customization and the ability to integrate Katana into your pipeline has always been very important to Katana's philosophy. Our design processes try to ensure we provide extensibility through APIs and other such mechanisms. In Katana 1.x however, there were certain limitations as to how you, as a facility, could extend the Katana node graph. It was not possible for example, to write your own equivalent to the Merge node. In Katana 2.0 we are proud to introduce the new Op API.

## The Op API

The Op API supersedes the Scene Graph Generator (SGG) and Attribute Modifier Plug-in (AMP) APIs previously used to write new types of nodes for Katana. Both of these APIs have their own limitations and were not used internally within Katana. In Katana 2.0, the Op API, which we use to write all of the operators used by the core Katana nodes, are now exposed directly to you.

The main benefit of being able to use the Op API is that you can now write Ops that can both see the incoming scene graph from multiple inputs, and create new locations. This can be great for:

- Crowds
- Instancing
- Advanced merges
- Context-aware generators/importers

The code you need to write is also much simpler.

NOTE: For more information on how Ops work and their development, see the *Op API* chapter of the *Katana Technical Guide*.

Fear not though! To ease the transition to Katana 2.0, almost all existing Scene Graph Generators and Attribute Modifier Plug-ins can still be used in Katana 2.0.

Custom Ops can be sourced from the **Ops** sub-directory of any **KATANA_RESOURCES** path.

## Customizing Keyboard Shortcuts

Custom keyboard shortcuts for certain actions and key events in the UI can now be defined in a configuration file stored in your Katana folder: `$HOME/.katana/shortcuts.xml`. The current assignments can be viewed in the **Keyboard Shortcuts** tab.

NOTE: For more information on the shortcuts.xml file and its creation see the *Managing Keyboard Shortcuts and the shortcuts.xml File* chapter of the *Katana Technical Guide*.

## SceneGraphView Widget

The new SceneGraphView widget is part of the Katana UI APIs and provides functionalities to implement custom views of the scene graph. In Katana 2.0, this widget type is used in the **Scene Graph** tab and for the object table of GafferThree nodes in the **Parameters** tab. You can make use of the SceneGraphView widget in your own tab plug-ins and SuperTools if you need to show a view of scene graph data.

Main features of the SceneGraphView widget include the following:

- View the scene graph produced at an arbitrary node.
- Customizable columns allowing you to view attributes at any scene graph location directly in the SceneGraphView.
- Ability to specify arbitrary scene graph locations to act as top-level locations from which to view the currently generated scene graph.
- Ability to customize context menus based on the scene graph location, column and/or current selection state when they are opened.
- Ability to interrogate attributes at arbitrary locations in the scene graph and be notified through a callback when they have changed.
- Support for drag-and-drop of items through custom event callbacks.
- Support for changing parameter values of multiple selected items in SceneGraphView widgets. When changing the value in a cell while multiple rows are selected, the cells in other selected rows in the same column change as well.
- Support for indicating a difference between the final value of an attribute that corresponds to a cell in a SceneGraphView widget and the value of a parameter that corresponds to the same cell, using an asterisk shown next to the value within a cell. When moving the pointer over such a cell, a pop-up widget is shown, showing the value of the attribute.

NOTE: You can find example tab plug-ins that demonstrate the use of the SceneGraphView class in: `$KATANA_HOME/plugins/Src/Resources/Examples/Tabs/PrototypeSceneGraphTab.py`.

## Keyboard Shortcut Manager

The `UI4.App.KeyboardShortcutManager` Python module has been added for registering action callbacks to which keyboard shortcuts can be assigned, as well as key press callbacks and key release callbacks.

Presently, this mechanism only covers the following areas of the UI:

- The buttons next to the main menu in Katana's main application window:
    - Shelf Actions

    - Flush Caches
    - Toggle Scene Graph Implicit Resolvers
    - Render Only Selected Objects
    - Auto-key All

- The **Scene Graph** tab.
- The GafferThree object table of GafferThree nodes that are edited in the **Parameters** tab.
- Custom tab plug-ins that derive from `UI4.Tabs.BaseTab`.

You can view the currently assigned keyboard shortcuts of actions and key events registered with the keyboard shortcut manager in the **Keyboard Shortcuts** tab. Katana 2.0 allows customizing keyboard shortcuts for those registered actions and key events (see Customizing Keyboard Shortcuts).

## LiveRenderAPI

The new `LiveRenderAPI` Python package provides access to several functions that allow you to modify the behavior of the Live Rendering system. It contains the following functions:

- `SendCommand()` - Sends custom Live Render commands to the renderer plug-in through the command socket.
- `SendData()` - Sends custom data updates to the renderer plug-in through the data socket.
- `AppendTerminalOp()` and `RemoveTerminalOp()` - Adds or removes additional terminal Ops to the Live Rendering client allowing you to customize the scene graph data that is passed through to renderers.
- `InsertTerminalOp()` - Inserts a terminal Op into the Live Rendering client at a specified position index.
- `GetTerminalOps()` - Returns a list of tuples describing each terminal Op along with its Op args.
- `ClearAllTerminalOps()` - Removes all Live Rendering terminal Ops, including the defaults (specified in renderer info plug-ins).
- `RestoreDefaultTerminalOps()` - Restores the default terminal Ops and removes all others.
- `InsertTerminalOp()` - Inserts a terminal Op into the Live Rendering client at a specified position index.
- `GetTerminalOps()` - Returns a list of tuples describing each terminal Op along with its Op args.

## Layered Menus in the Node Graph Tab

The new `LayeredMenuAPI` allows you to define and register custom menus for the **Node Graph** tab that appear and behave in the same way as the built-in node creation menu that is shown when pressing the **Tab** key. These custom menus are named *layered menus*, as they appear in a menu layer on top of the nodes that make up the node graph of a Katana project.

The entries that are shown for a layered menu can be customized with arbitrary text and a color per entry. When entering text while a layered menu is shown, its entries are filtered based on the entered text, just like entries of the node creation menu are filtered when entering the name of a node type.

When an entry from a layered menu is chosen, custom Python code can be executed. One common use case for layered menus is the ability to create and configure nodes of certain types based on a pre-defined or dynamic list of menu entries for you to choose from.

An example script that registers a layered menu for the **Node Graph** tab, which shows the names of available PRMan shaders and creates a PrmanShadingNode node with the chosen shader set on it when one of the menu entries is chosen, can be found in the following location:

```
$KATANA_HOME/plugins/Src/Resources/Examples/UIPlugins/CustomLayeredMenuExample.py
```

# 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 (equivalent to Geolib2 Attribute Modifiers).

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.

## Main Differences Between Geolib2 and Geolib3

- Geolib2 does 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 is traversed using an implicit index mechanism, for instance `getFirstChild()` and `getNextSibling()`, with scene graph location names determined by the **name** attribute. In Geolib3, child locations are natively indexed by name. Therefore, in Geolib3 you can selectively cook a location, by name, without cooking any peers. Consequently, the **name** attribute is meaningless. This also implies that locations cannot rename themselves, you can rename children however.
- Geolib2 is not amenable to either asynchronous or concurrent evaluation. Geolib3 supports both of these modes of operation.

NOTE: For more details on the above topics, see the *Graph State Variables* chapter of the *Katana User Guide* and the *Porting Plug-ins*, *Op API*, and *NodeTypeBuilder* chapters of the *Katana Technical Guide*.

# Important Changes

## Nodes

### AttributeScript

- `GetAttr("name")` no longer accesses the location name, since location names are no longer simple string attributes. You should use `GetName()` instead.
- `SetAttr("name", <newName>)` no longer renames a location. Locations are explicitly named on creation in Katana 2.x. The Rename node or OpScript nodes should be used instead.
- `GetAttr(<attrName>, inherit=True)` now consistently returns an attribute from the input to the AttributeScript node, even when queried at other locations. In previous versions, in some situations, the output of the AttributeScript node would be considered.
- `GetChildNames(atLocation=<locationPath>)` causes the script to abort if the requested location is not an ancestor of the location the AttributeScript is operating upon. The script restarts from the beginning once the location is cooked. In most cases this is inconsequential, but can have an impact if the script is dealing with external resources (database, filesystem, and so on).
- Functions which usually return a `ScenegraphAttr`, for instance, `GetAttr(<attrName>, asAttr=True)` now take the optional `asFnAttribute` parameter (default: `False`) to return the newer `FnAttribute` type. There are a number of method differences between `FnAttribute` and the old `PyScenegraphAttr` type, including the following:
    - `FnAttribute.Attribute` has no `type()` method, instead use `isinstance(attr, FnAttribute.<Type>Attribute)`
    - `FnAttribute.GroupAttribute` has no method `childNames()`, instead use `childList()`

NOTE: AttributeScripts can now use `FnAttribute` instead of `ScenegraphAttr`, for more information see the *ScenegraphAttr Porting Guide* of the *Katana Technical Guide*.

- The following methods to resolve and query information about asset IDs are now available in AttributeScript nodes through the `Util` module:

```
DefaultAssetPlugin.isAssetId(string)
DefaultAssetPlugin.containsAssetId(string)
DefaultAssetPlugin.resolveAsset(assetId)
DefaultAssetPlugin.resolvePath(path, frame)
DefaultAssetPlugin.getUniqueScenegraphLocationFromAssetId(assetId, includeVersion)
DefaultAssetPlugin.getRelatedAssetId(assetId, relation)
```

- In order to support the new threading models in Katana 2.x and avoid UI blocking, AttributeScripts are now evaluated by a separate pool of Python interpreters. This is to mitigate the limitations the CPython GIL imposes. Consequently, the following considerations should be taken into account:
    - Setup scripts can be run more than once, but only once per interpreter in the pool.
    - Child locations may not run in the same interpreter as parent locations.

It is important that any code making use of the user module does not assume that it is the same instance as was present when the script ran in another location.

- As AttributeScripts are now run through an interpreter pool (see above), simple AttributeScripts now have a greater performance impact than in previous Katana versions (due to the setup/IPC overhead). As such, it's recommended to use OpScript for anything that isn't using pymath or any heavy lifting through bindings to third-party libraries, as it doesn't include the same overheads.

NOTE: For more information on Python interpreter processes, see the *Python Processes and Geolib3* chapter of the *Katana Technical Guide*.

## Gaffer

- The existing legacy Gaffer node type from Katana 1.x is still present, and previously created projects should continue to work, but it is advisable to move to using the new GafferThree node type where possible.
- Gaffer nodes from 1.x projects are updated to 2.0-compatible Gaffer nodes by an update script.
- The way that Sky Dome items are implemented in classic Gaffer has changed. Instead of an **arnoldSurfaceShader** of type **skydome_light** on the item's Material node, materials on Sky Domes are now resolved internally in Gaffer.

## Alembic_In

The Alembic_In node type now supports a **useOnlyShutterOpenCloseTimes** parameter that forces the Alembic cache to only use the time samples corresponding to shutter open and close times when the **maxTimeSamples** option is set to **2**.
The parameter is available in the **advanced** section, in the **Parameters** tab.

NOTE: The **useOnlyShutterOpenCloseTimes** argument is also supported by the AlembicIn Op.

## VelocityApply

The following parameters have been added to VelocityApply nodes:

- **velocityAttribute** - 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 to be used to interpret the values stored in the velocity attribute, with the following options:
    - **units / second**
    - **units / frame**

## ScenegraphGeneratorSetup

- The Alembic_In Scene Graph Generator has been removed, having been superseded by the AlembicIn Op.
- The signature of the `createProxyAttr()` method of `BaseProxyLoader` has been modified to return a GroupAttribute that sets up the execution of an Op instead of a Scene Graph Generator. This Op is specified by the child **opType** StringAttribute and optional child **opArgs** GroupAttribute. Proxy resolution using a custom Scene Graph Generator is supported by setting the **opType** as **ScenegraphGeneratorHost** and passing the name of the generator as the StringAttribute **opArgs.generatorType**, and the optional args in GroupAttribute **opArgs.args**.

## APIs

- Attribute Modifier Plug-ins (AMPs) and Scene Graph Generator plug-ins (SGGs) need to be recompiled. For minor header/source changes, see the *Porting Plug-ins* chapter of the *Katana Technical Guide*.
- AMPs are no longer resolved with an AttributeModifierResolve node. They are resolved with an OpResolve node.
- AMPs can no longer rename locations (through `setAttribute("name", "newName")` or otherwise). Locations are explicitly named on creation in Katana 2.x. The Rename node or OpScript nodes should be used instead (or the core Op API).
- The **name** attribute is no longer used to determine the name of a scene graph location. Querying the attribute only returns any data set by other calls to `setAttribute()`/`SetAttr()`, and not the name of the location, which means that **name** is just like any other attribute. In AMPs, the current location's name can be queried with `AttributeModifierInput::getName()`, and in AttributeScript, with `GetName()`.
- Python-based AssetAPI plug-ins are known to be a performance bottleneck due to the overhead of executing Python code in separate processes. This is particularly prominent with the high numbers of calls to `isAssetId()` that are executed during material resolve on shader parameters. It is therefore advisable to use C-based implementations where possible.

NOTE: For more information on Python interpreter processes, see the *Python Processes and Geolib3* chapter of the *Katana Technical Guide*.

- Python-based Render Location plug-ins have been removed due to the inherent performance bottleneck of executing Python code when evaluating `/root`. This is a special case that should be kept as lightweight as possible, due to the frequency of evaluation in interactive sessions. The core plug-ins have been replaced by C++

RenderOutputLocation plug-ins that are shipped as source and can be found in `plugins/Src/RenderOutputLocations`.

- Across the board, `ScenegraphAttr` has been replaced by `FnAttribute`, with a new, optimised implementation.
- `FnAttribute::GroupBuilder::build()` has been modified to support an optional `builderMode` parameter to define if the content of the builder has to be retained or flushed when the resulting GroupAttribute is created. The `builderMode` parameter has type `BuilderBuildMode` and supports only the values `BuildAndFlush` and `BuildAndRetain`:

  ```
  GroupAttribute build(BuilderBuildMode builderMode = BuildAndFlush);
  ```

  Notice that, in Katana 2.0, `BuildAndFlush` is the default value for the `builderMode` parameter, so `GroupBuilder::build()` flushes the content of the builder by default.

  The same behavior applies in the Python and LUA bindings for `FnAttribute::GroupBuilder`.

  Calling `GroupBuilder::build()` on the same `GroupBuilder` multiple times then results in a valid `GroupAttribute` being returned only for the first invocation while, for the following ones, an invalid `FnAttribute` is returned.

  Instead of calling `FnAttribute::GroupBuilder::build()` multiple times, the `GroupAttribute` returned by `build()` can be stored in a variable and used in different places in the code. Alternatively the `BuildAndRetain` value for the `builderMode` parameter can be used.

## Viewer Proxies

Ops can now 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.
- **Op-based** - Ops can be chained to create the geometry to be used as a proxy by adding group child attributes to the **proxies.viewer** group attribute on the target location. Each child group attribute represents an Op and its content must contain:
  - a string attribute named **opType** defining the type of the Op to be used.
  - a group attribute named **opArgs** containing attributes defining the Op arguments.

  Here's an example of the attributes hierarchy using two Ops to generate the proxy geometry:

```
Location
    /root/world/geo/group

Attributes:
   ...
   proxies
     viewer
       proxyOp_1
         opType 'AlembicIn' (StringAttribute)
```

```
        opArgs
           fileName '/tmp/myProxy.abc' (StringAttribute)
      proxyOp_2
        opType 'Messer' (StringAttribute)
        opArgs
          displacement 0.23 (DoubleAttribute)
...
```

Proxy caches are considered animated by default. Static proxy caches can be defined by setting the **proxies.static** IntAttribute to **1**.

## Attribute History

There is a new API for querying Attribute History from Python. You can find it in the `UI4.Util.AttributeHistory` module.

Attribute History can be queried synchronously, in which case the UI blocks until the result is computed and returned, or asynchronously if you provide a callback to run when the computation is complete.

## Handling of Font Preferences

Katana 2.0 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.

## Documentation

The **Help** tab has been deprecated in favor of serving HTML documentation in your web browser. The documentation is now generated by Sphinx, which features a number of niceties, such as searching and syntax highlighting.

The **Examples** page of the previous HTML documentation has been replaced by a dedicated **Example Projects** tab. The tab can be launched through Katana's main menu bar, by navigating to **Help** > **Example Projects**.

## Changes in Third-Party Library Dependencies

The changes in the third-party library dependencies are the following:

- **Alembic 1.5.3** - provides better support for the Ogawa data storage backend.
- **OpenColorIO** - Katana previously shipped with a build of OpenColorIO that used `FnOpenColorIO` namespaced symbols, but the OpenColorIO libraries were not named accordingly (with the Python bindings

`libOpenColorIO.so` and `PyOpenColorIO.so`). This caused problems with using a custom facility-installed OpenColorIO in parallel with the Katana libraries. This has been updated so that the libraries shipped with Katana are now Fn-prefixed too (`libFnOpenColorIO.so` and `FnPyOpenColorIO.so`). Using the Python bindings is still possible through `import FnPyOpenColorIO` and any code using `PyOpenColorIO` needs updating to either use `FnPyOpenColorIO`, or a facility-installed OpenColorIO could be used instead of the one that ships with Katana.

- **Python 2.7.3** - upgraded to match the VFX Platform CY2014 specification
- **Qt 4.8.5** - upgraded to match the VFX Platform CY2014 specification