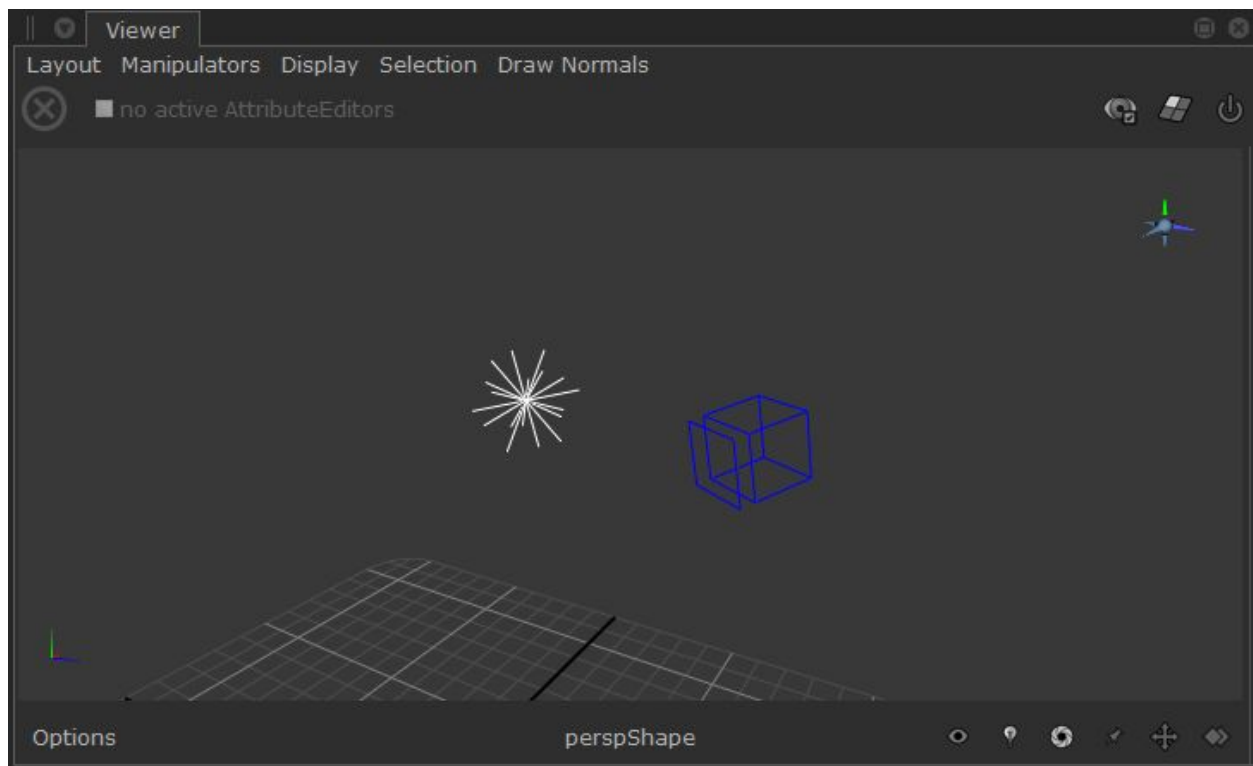


Katana Viewer API: Viewer Modifier Porting Guide

Last updated 19th April 2018

Summary

Viewer Modifier plug-ins are a feature of Katana's legacy, OpenSceneGraph-based **Viewer** tab, which allows custom drawing code to be executed for arbitrary scene graph location types. Viewer Modifiers are typically used to draw viewer-only representations of light and camera locations.



Viewer Modifiers are limited in their power and flexibility for various reasons, so with the development of the new Viewer API an alternative solution (now known as Locators) was developed. This guide will cover an approach for emulating the behaviour of legacy Viewer Modifier plug-ins, and how to convert any existing Viewer Modifiers to the new Viewer API's Locators.

ViewerPluginExtensions

A new type of Python plug-in called a `ViewerPluginExtension` has been added which contains several callback functions that are triggered from the `BaseViewerTab` class in response to certain events. These functions are `onTabCreated()`, `onDelegateCreated()`, `onViewportCreated()` and `onApplyTerminalOps()`. Each function is passed relevant arguments, such as the `ViewerDelegate` or `ViewportWidget` objects. This allows a `ViewerPluginExtension` to modify a viewer's behaviour by applying other plug-in types which may need to work together, such as `ViewerDelegateComponents` and `ViewportLayers`.

An example of a VPE may look like this:

```
from PluginAPI.BaseViewerPluginExtension import BaseViewerPluginExtension

class BallViewerPluginExtension(BaseViewerPluginExtension):
    def onDelegateCreated(self, viewerDelegate, pluginName):
        viewerDelegate.addComponent('BallComponent', 'BallComponent')

    def onViewportCreated(self, viewportWidget, pluginName, viewportName):
        # Insert a layer named BallLayer at index 4 into the given viewport widget
        viewportWidget.insertLayer('BallLayer', 'BallLayer', 4)

PluginRegistry = [
    ("ViewerPluginExtension", 1, "BallViewerPluginExtension", BallViewerPluginExtension),
]
```

Viewport Layer & Viewer Delegate Component

These are two separate plug-ins that can be added to a viewer, usually by a `ViewerPluginExtension`. The `ViewerDelegateComponent` is used to track the scene graph and identify locations that require custom drawing and the `ViewportLayer` is responsible for drawing the locations that have been identified. These classes give you complete control over the drawing of your own geometry, but will require some bookkeeping in order to track changes to the scene graph and selection states.

Typically a viewer will have one `ViewerDelegate`, but possibly multiple viewports, so computationally heavy work is best suited in the VDC. Any code that requires a current OpenGL context should be called within one of the relevant `ViewportLayer` functions, or you can call `Viewport::makeGLContextCurrent()` before executing.

An example of this style of plug-in is distributed with Katana and can be found at:

```
$KATANA_ROOT/plugins/Src/ViewerPlugins/BallLocationType
```

Once you have a `ViewportLayer` and `ViewerDelegateComponent` plug-in you need to write a `ViewerPluginExtension` as seen above to append them to the `ViewerDelegate` and `Viewport`.

Locator Plug-ins

In order to accelerate the process of converting old `ViewerModifier` plug-ins for use in a Viewer API based viewer, we have provided a base implementation of a `ViewportLayer` and `ViewerDelegateComponent` pair and distribute their source code with Katana. The header for these plug-ins can be found in `<KATANA_HOME>/plugin_apis/include/FnViewer/utils/FnBaseLocator.h`. There is also an example that uses these classes to draw a spotlight cone in a similar manner to how you might have done in an older `ViewerModifier`. You can find the example in `<KATANA_HOME>/plugins/Src/ViewerPlugins/ExampleSpotlightLocator/`.

In order to get started you simply have to write a class which derives from `FnKat::ViewerUtils::FnBaseLocator` which is then registered with the `FnBaseLocatorVDC` class via the `REGISTER_VMP()` macro.

Each locator needs to have four static functions before it can be successfully registered with the `ViewerDelegateComponent`. These are as follows:

```
/// Determines whether this locator plug-in should be used to draw the
/// location described in the given viewer location event.
static bool matches(const FnKat::ViewerAPI::ViewerLocationEvent& event);

/// Specifies whether this locator plug-in overrides the standard geometry
```

```

/// representation for matched locations.
static bool overridesBaseGeometry(const FnKat::ViewerAPI::ViewerLocationEvent& event);

/// Gets the bounds of the given location.
static FnAttribute::DoubleAttribute getBounds(
    const FnKat::ViewerAPI::ViewerLocationEvent& event);

/// Computes the extent of the given location.
static FnAttribute::DoubleAttribute computeExtent(
    const FnKat::ViewerAPI::ViewerLocationEvent& event);

```

The VDC will then use the `matches()` function of all Locators that have been registered with it to determine whether it should be used to draw each location. This function is similar to the same function found in `ViewerManipulators`, in that it allows your plug-in to match the location based on any conditions, rather than just the location type, as was necessary with old style `ViewerModifiers`. This implementation also allows multiple Locators to be active for a single location, which was not previously possible. This interface should allow you to convert old viewer modifiers into `Locator` plug-ins with relative ease. If more control over the plug-in behaviour is required, it is possible to write a class that derives from the base implementations of the `VDC` and `ViewportLayer` that we provide. A simple example of this is included in the `ExampleSpotlightLocator` plug-in

Please note that the `VDC` and `ViewportLayer` classes should be compiled into the same dynamic library as your `FnKat::ViewerUtils::FnBaseLocator` classes to ensure that you do not encounter issues related to passing complex objects across the boundary of a dynamic library.

Copyright © 2018 The Foundry Visionmongers Ltd.