Custom Katana Ops

Configuring RfK with Custom Ops

In order to configure RfK to use custom Ops you must build a configuration file to specify your Ops and the order in which they should be evaluated. The file:

```
$KATANA_RESOURCES/RfK/config.xml
```

is read from any location relative to an entry in the KATANA_RESOURCES path. The results are combined in the order read.

The configuration currently supports specification of additional "terminal Op" instances added onto the scene as visible to rendering and live render update observation.

The config.xml file format is:

```
<rfkConfig format="1">
    <!-- Ops which affect the scene as seen at ribgen time -->
    <renderTerminalOps>
        <op type="SomeOpName" insertWhen="beforeVstructExpansion"/>
        <op type="AnotherOpName" insertWhen="last" needsSystemArgs="true"/>
        <op type="AnOpWithArgs" insertWhen="last">
            <attr type="GroupAttr">
                <attr name="taco" tupleSize="1" type="IntAttr">
                    <sample size="1" time="0" value="1 "/>
                </attr>
                <attr name="cheese" tupleSize="1" type="StringAttr">
                    <sample size="1" time="0">
                        <str value="salsa"/>
                    </sample>
                </attr>
            </attr>
        </op>
   </renderTerminalOps>

   <!-- Ops which are evaluated during live render updates -->
   <liveRenderTerminalOps>
        <op type="SomeOpName" insertWhen="beforeVstructExpansion"/>
   </liveRenderTerminalOps>
</rfkConfig>
```

Ops can be added as either renderTerminalOps or liveRenderTerminalOps. The format of the Ops within each type are the same, the only difference is when the Ops are evaluated by Katana.

`renderTerminalOps`

The "op" elements added here affect the scene as seen at render time.

`liveRenderTerminalOps`

The "op" elements here are added to the live render observer within the Katana process. If you need to manipulate the scene in any manner unique to live render updates, you may do this here.

An "op" element has two required properties:

- type a string containing the name of a registered Op type = insertWhen a string indicating where among the terminal ops added by RfK itself the op should be inserted. See table below for a list of valid string values.

An "op" element may have these optional properties:

- needsSystemArgs values of "yes", "true" or "1" result in the "system" GroupAttribute being added to the Op's arguments. This contains information about the current frame and shutter information.

An "op" element may also have a child element of type "attr". This is expected to be the XML serialization of an FnAttribute::GroupAttribute, which can be generated from Python or C++ via the attr's getXML() method.

Valid insertWhen Options

Option	Description
first	Before all other terminal Ops
beforeVstructExpansion	Before virtual struct connections are expaned (i.e. before the Op PRManResolveVirtualStructConnections is called)
afterVstructExpansion	After virtual struct connections are expaned (i.e. after the Op PRManResolveVirtualStructConnections is called)
last	Append your Op after all others.

Supported Configuration Options

In addition to the renderTerminalOps and liveRenderTerminalOps hooks there are several other specialized options that are recognized by the RfK plugin. The first two are a simple configuration/extension options. The last two enable the passage of non-standard information from args files to custom terminal Ops.

shaderPathRecursionLimit
Searching through the shader search paths will, by default, recurse down 3 levels before terminating the search in that path. This is configurable through the shaderPathRecursionLimit option in config.xml. Setting the value to 0 (zero) will disable recursion when searching the shader paths.

Example entry in config.xml:

```
<!-- Turn off shader search path recursion
<int name="shaderPathRecursionLimit" value="0"/>
```

shaderPostProcessFunctionName
A shader's "info" attribute can be modified using a function to run after a shader has been loaded and configured. The shaderPostProcessFunctionName refers to an attribute function which takes a group attribute in (the existing info attribute) and returns a new info attribute.

Example entry in config.xml:

```
<int name="shaderPostProcessFunctionName" value="MyFunc"/>
```

hintPassthroughNames
Non-standard parameter hints can be passed from args file through to an Op using shader parameter info mechanism (PRManAddShaderParameterInfoOp) and the hintPassthroughNames list.

outputTagPassthroughNames
Non-standard per-shader, per-output tags can be passed from args file through to an Op using shader parameter info mechanism (PRManAddShaderParameterInfoOp) and the outputTagPassthroughNames list.