Args File Reference

This document focuses on Katana usage as a practical example.

What Is An Args File?

Args files are XML files that provide the user interface for shader parameters. RenderMan for Katana looks for Args files in one of the following directories:

1. Args subdirectory of the shader directory. This is recommended.
2. ../Args directory relative to the shader directory.
3. Args subdirectory of $KATANA_RESOURCES.

Args files are **required for plugin (C++) shaders**; they are the only source for information to the DCC about the shader parameter types, defaults, etc. (Note, however, that RenderMan itself does not read the Args file; it gets this information from the parameter table of the shading plugin). The name of the Args file must have the same base name as the shader, e.g. PxrBump.args for PxrBump.so, PxrRectLight.args for PxrRectLight.so, etc. The order of the shader parameters in the Args file represents the order in which they will appear in the user interface.

How To Create An Args File

An Args file is an ASCII file that can be created manually using any text editor.

Note: If you are using RenderMan for Katana some aspects the user interface can be edited interactively then exported to the Args file. Refer to the documentation in your Katana installation for more details ($KATANA_HOME/docs/pdf/ArgsFiles.pdf).

Args File Format

The basic format for an args file is as follows:

```
<args format="1.0">
    <param name="someParameter"/>
    <param name="anotherParameter">
</args>
```

Shader Type

PRMan extension

The type of shader is specified using the shaderType element. This is necessary for plugin shaders as there is no mechanism by which to query the shader type. An example for a pattern plugin would be:

```
<shaderType>
    <tag value="pattern"/>
</shaderType>
```

Basic Elements

param

Elements of type param describe the presence and order of parameters within the shader UI. Each param element is required to have a name property; all other properties are optional. See below for a detailed list of param properties.

page

Parameters may be grouped into pages via page properties, either as a stand-alone page or nested via dot-delimited values:

```
<param name="someParameter" page="Lighting" />
<param name="anotherParameter" page="Lighting.Advanced"/>
```

Page grouping may also be specified via enclosing page elements:

```
<page name="Lighting">
    <param name="someParameter"/>
    <page name="Advanced">
        <param name="anotherParameter"/>
    </page>
</page>
```

Parameters within a page group are collected in the UI under a group widget pane that can be "open" or "closed". By default the group is closed but this behavior can be modified using the open property on the page element:

```
<page name="Advanced" open="True">
    <param name="anotherParameter"/>
</page>
```

help

Embedded documentation may be specified with either a property or a child element of type help:

```
<param name="someParameter" help="This parameter increases render time."/>
```

When using a child help element, its contents are interpreted as XHTML for more sophisticated formatting:

```
<param name="anotherParameter">
    <help>
        <p>
        This parameter is <b>very</b> important.
        </p>
        <i>Please be careful when setting it.</i>
    </help>
</param>
```

Both forms are also supported on args and page elements. For more details on embedded documentation and other features of the help element please see the Args File UI Hinting document.

Comments

Comments are the same as they are in HTML and XML:

```
<!-- Useful commentary goes here. -->
```

Elements and Properties

Args files were originally created as a way of providing UI hinting for shaders in Katana. PRMan has since adopted and extended this format. Katana's set of elements, parameters and tags are described in detail in the Katana documentation and in the Args File UI Hinting document. Below is a summary of the most commonly used components for the built-in PRMan shaders. Portions that have been adopted specifically for PRMan and RenderMan for Katana and Maya are noted.

Parameter Definition

Details about each param element can be specified using the following properties:

Parameter Properties

Property	Description	Requirement	Origin
`name`	Specifies the exact shader parameter.	Required	Katana
`label`	String to display in the UI for the parameter. If no label is specified the name of the parameter is used.	Optional	Katana
`type`	The parameter type. Recognized types are: *float*, *int*, *string*, *color*, *normal*, *point*, *matrix*, *vector*, *struct*, *bxdf, lightfilter, displayfilter, and samplefilter*.	Required for plugin shaders in RfM.	PRMan extension
`default`	Specifies default parameter value. For color, point, normal, and vector, three values are required. For array size > 1, default values are separated by space. Examples: type = "float" default = "5.0". type = "color"", default="0.18 0.18 0.18" type = "normal" default="0 0 0" type = "float" arraySize = "5" default = "15.0 25.0 8.0 5.0 40.0" type = "color" arraySize = "2" default = "0.18 0.18 0.18 0.18 0.18 0.18" For dynamic array, since the array size is unknown, specifying the default values will not work.	Required for plugin shaders	Katana
`arraySize`	Specify the array size. Default values are separated by spaces. `<param name="resolution" type="float" arraySize="2" default="1024 1024"/>`	Optional	Katana
`isDynamicArray`	Set to "1" if the parameter is a dynamic array. Note that default for dynamic array must not be specified. `<param name="layers" isDynamicArray="1" arraySize="-1" widget="dynamicArray"/>`	Optional	Katana
`widget`	The widget property specifies which user interface widget should used for a parameter. If no widget type is defined the widget will default to a numeric type. The standard widget types are defined below.	Optional	Katana

The following table is a list of the commonly used widget types for RfM and RfK. There are several other widget types defined by Katana which are recognized by RfK but they are not currently recognized by RfM. The table includes a list of commonly-used hints for each widget types. Please see Args File UI Hinting for detailed explanations of these and other available hints.

List of Common Widget Types

Widget Type	Tag Name	Description	Common Hints
Default	`default or none`	Each parameter type has a default widget type. If the widget type is not specified in the args file it will be determined by the shader parameter data type.	None
Numeric	`int`	Default widget used for numeric parameter types. Common hints are listed below.	`min, max, sensitivity, int slider, slidermin, slidermax`
String	`string`	Default widget for string parameters.	`replaceRegex, replaceWith`
Boolean	`boolean`	A dropdown list with "Yes" and "No" options.	None
Check Box	`checkBox`	Boolean parameter displayed as a check box rather than a pop-up.	None
Popup /Selector	`popup`	A pop-up menu or combo box with literal choices for paramter values.	`options, editable`
Mapping Popup	`mapper`	A pop-up menu with associative choices. Each value has an associated display value.	`options, hintdict`
Color	`color`	A color parameter where a color picker will be available to pick a color. Note that in Katana, the color picked will return a *linearized* RGB value.	None
File Input	`fileInput`	A file parameter where a file browser icon will be available to pick a file.	`typefilter, presets, fileType`. In addition RfM supports the `options="texture"` option for automatically running txmake.
Null /Hidden	`null`	Hide the parameter from the user interface.	None
Float Ramp	floatRamp	Float ramp editor. Require_Knots, _Floats, and _Interpolation for the parameter. See example below.	None
Color Ramp	colorRamp	Color ramp editor. Require _Knots, _Colors, and _Interpolation for the parameter. See example below.	None
Asset Input	assetIdInput	An asset browser for selecting an input. Use with options="texture". See example below.	None

Widget Type Examples

```
<param name="diffuseColor" widget="default"/>

<param name="channel" type="string" widget="popup" options="R|G|B|A"/>

<param name="stops" label="Stops" type="float"
       default="0.0" min="-10" max="10"
       slider="True" slidermin="-4" slidermax="4"
       widget="float">

<param name="atlasStyle" type="int" default="0" widget="mapper">
    <hintdict name="options">
        <string name="None"   value="0"/>
        <string name="MARI"   value="1"/>
        <string name="Mudbox" value="2"/>
    </hintdict>
</param>

<param name="filename"
       label="Filename"
       type="string"
       default=""
       widget="assetIdInput"
       options="texture"
       connectable="False">
</param>

<page name="Color Ramp" open="False">
        <param name="colorRamp"
               label="Color Ramp"
               type="int" widget="colorRamp" default="4"
               colorBarHeight="25" restrictComponents="True">
        </param>
        <param name="colorRamp_Knots"
               type="float"
               widget="null"
               default="0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1" arraySize="16"/>
        <param name="colorRamp_Colors"
               type="color"
               widget="null"
               default="1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1"
               arraySize="16"/>
        <param name="colorRamp_Interpolation"
               type="string"
               default="linear"
               widget="null" options="linear|catmull-rom|bspline|constant"/>
 </page>

<page name="Falloff" open="False">
        <param name="falloff"
               label="Fall Off"
               type="int" widget="floatRamp" default="6"
               connectable="False"
               bgColor="0.25 0.25 0.25" underSplineColor="0.5 0.5 0.5">
            <help>falloff: Controls the transition from the core to the edge.</help>
        </param>
        <param name="falloff_Knots"
               type="float"
               widget="null"
               default="0 0 0.3 0.7 1 1 1 1 1 1 1 1 1 1 1 1"
               arraySize="16"/>
        <param name="falloff_Floats"
               type="float"
               widget="null"
               default="0 0 0.2 0.8 1 1 1 1 1 1 1 1 1 1 1 1"
               arraySize="16"/>
        <param name="falloff_Interpolation"
               type="string"
```

```
               default="bspline"
               widget="null" options="linear|catmull-rom|bspline|constant"/>
</page>
```

Input and Output Tags

Katana requirement: All input and output tag values must be all lowercase.

Each shader can define a set of outputs in the args file. In addition each parameter of a shader can define what types of input are allowed.

Output connections are specified using the output element with a series of tags enumerating the output types:

```
<output name="resultRGB">
    <tags>
        <tag value="color"/>
        <tag value="diffuse"/>
    </tags>
</output>
```

Output tag values can be any string, but in order for Katana to recognize them as valid outputs the value string must be all **lowercase** and each tag value must be specified on a separate line.

Input connections are defined as a tags child element of param:

```
<param name="inputDiffuse" type="color" widget="default">
    <tags>
        <tag value = "diffuse"/>
    </tags>
</param>
```

Tag values can be any valid output connection string. Boolean logic is also available for more advanced rules about input connection validity:

```
<param name="inputDiffuse" type="color" widget="color">
    <tags>
        <tag value = "(diffuse and color) or struct"/>
    </tags>
</param>
```

Connections are disabled by setting connectable="False"

```
<param name="invertT"
       type="int"
       widget="checkBox"
       connectable="False">
</param>
```

Please see the Katana Technical Guide section on Typed Connection Checking for more details about input/output tags and input connection boolean logic.

Parsing Args Files

PRMan uses RapidXML for parsing Args files. See $RMANTREE/lib/examples/args/parse/parseArgs.cpp for an example of a standalone C++ program to parse an .args file. To build this program on Linux, for example, run:

```
make -f Makefile.linux
parseArgs $RMANTREE/lib/plugins/Args/PxrChecker.args
```

parseArgs will output a list of parameters and their attributes that are supported in RenderMan for Katana and Maya.

Sample Args File

The PxrChecker.args file (for PxrChecker.so shader) is provided as an example, to provide more information about the structure of the file:

```
<args format="1.0">
    <shaderType>
        <tag value="pattern"/>
    </shaderType>
    <help>A checker pattern</help>
    <param name="colorA"
        label="Color A"
        type="color"
        default="1.0 1.0 1.0"
        widget="color">
    </param>
    <param name="colorB"
        label="Color B"
        type="color"
        default="0.0 0.0 0.0"
        widget="color">
    </param>
    <param name="manifold"
        label="Manifold"
        type="struct"
        default=""
        widget="default">
        <tags>
            <tag value="struct"/>
            <tag value="manifold"/>
        </tags>
    </param>
    <output name="resultRGB">
        <tags>
            <tag value ="color"/>
            <tag value ="vector"/>
            <tag value ="normal"/>
            <tag value ="point"/>
        </tags>
    </output>
    <output name="resultR">
        <tags>
            <tag value ="float"/>
        </tags>
    </output>
    <output name="resultG">
        <tags>
            <tag value ="float"/>
        </tags>
    </output>
    <output name="resultB">
        <tags>
            <tag value ="float"/>
        </tags>
    </output>
    <rfmdata nodeid="1053270" classification="rendernode/RenderMan/pattern"/>
</args>
```