# Shader User Interface Hinting with Args Files

Args XML files are a means of hinting the user interface of shaders. The individual XML elements are also supported as embedded metadata on RSL parameters.

The basic structure of an Args file is:

```
<args format="1">
 <param name="someParameter"/>
 <param name="anotherParameter"/>
</args>
```

---

## Grouping with Pages

Elements of type **param** describe the presence and order of parameters within the shader UI. Parameters may be grouped into pages via **page** properties. Pages can be 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. This is often preferable as it provides a place to attach UI hints to pages themselves.  Nested **page** elements specify their **name** properties as a dot-delimited path relative to the enclosing value. The following is equivalent to the previous example.

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

Pages within Katana are implemented as groups with disclosure triangle controls. The default state of the disclosure triangle may be specified via an **open** property. Pages default to closed unless otherwise specified.

```
<page name="Lighting" open="true">
</page>
```

---

## Embedded Documentation

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. When attached to the top-level **args** element, they describe documentation for the entire shader.

```
<args format="1">
 <help>
  This shader shades things in myriad of useful ways. We really
couldn't live without it.
 </help>
 <page name="Lighting" help="The lighting controls are here.">
  <param name="someParameter"/>
  <page name="Advanced">
   <help>
    Please don't touch anything in here unless you know what you're
doing.
   </help>
   <param name="anotherParameter"/>
  </page>
 </page>
</args>
```

Additional emphasis may be placed on the documentation of a shader via **helpAlert** properties. These may have values of **normal** (the default if not specified), **warning** and **error**. In Katana, the icon which appears for a parameter containing documentation is colored yellow for **warning** and red for **error**.

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

## Widget Hint Types

Properties used to configure a widget are commonly called “hints.” Each hint is interpreted as a type — some of which may be expressed in multiple ways.

*Boolean* typed hints are considered true if they case-insensitively match against any of these values: “yes”, “y”, “true”, “1” or “on”. All other values are treated as false.

*Int*, *Float* and *String* typed hints are specified in common literal forms.

*List* typed hints may be specified either as a pipe-delimited (“|) string or as a child element of type **hintlist**. The following two forms are equivalent:

```
<param name=“someParam” alisthint=“a|b|c”/>
<param name=“anotherParam”>
 <hintlist name=“alisthint”>
  <string value=“a”/>
  <string value=“b”/>
  <string value=“c”/>
 </hintlist>
</param>
```

Child elements of **hintlist** elements may be of type **int**, **float** or **string** and may specify their value in either **value** or **default** properties.

*Dictionary* typed elements may be specified either as a pipe-delimited (“|”) of colon-separated (“:”) pairs or as a child element of type **hintdict**. The following two forms are equivalent:

```
<param name=“someParam” adicthint=“a:1.0|b:2.0|c:3.0”/>
<param name=“anotherParam”>
 <hintdict name=“adicthint”>
  <float name=“a” value=“1.0”/>
  <float name=“b” value=“2.0”/>
  <float name=“c” value=“3.0”/>
 </hintdict>
</param>
```

Child elements of **hintdict** elements may be of type **int**, **float** or **string** and may specify their value in either **value** or **default** properties. They must also include **name** property with a value unique amongst their peers.

Arrays or dictionaries may also represent entries in **hintlist** and **hintdict** elements by further nesting **hintlist** and **hintdict** elements.

## Common Widget Hints

The **widget** property specifies which user interface widget should used for a parameter. The standard widget types are defined in the next section. The following hints are applicable to all widget types.

**label**: supported types: *string*
Alters the display name of the widget. Please use this rarely as it can create confusion with regard to expressions and scriptability.

**readOnly**: supported types: *boolean*
If true, the value is not editable via the UI. This does not affect scripting.

**infoText**: supported types: *string*
Displays a line of text below the value of the widget. Please use this sparingly as the same mechanism is used for displaying error messages on some widget contexts

## Widget Types

### “number”

This is the default widget used for numeric parameter types. It supports these hints:

**min** *supported types: float, int*
Specifies a minimum value for when dragging on the widget’s label. Values entered via the text field are not clamped.

```
<param name=“someParameter” min=“0”/>
```

**max** *supported types: float, int*
Specifies a minimum value for when dragging on the widget’s label. Values entered via the text field are not clamped.

```
<param name=“someParameter” min=“0” max=“1.0”/>
```

**sensitivity** *supported types: float, int*
Specifies the label drag increment

```
<param name=“someParameter” min=“0” max=“1.0” sensitivity=“0.05”/>
```

**digits** *supported type: int*
Specifies the number of digits to display after the decimal point. The default value of -1 indicates full precision.

```
<param name=“someParameter” digits=“4”/>
```

**int** *supported type: boolean*
If true, floating point parameter types are displayed a integers. This affects display and also applies clamping on numeric entry.

```
<param name="someParameter" int="true"/>
```

**slider** *supported type: boolean*
When true, a visible slider is displayed to the right of the text entry field.

```
<param name="someParameter" slider="true" slidermin="-1.0"
slidermax="1.0"/>
```

**slidermin** *supported types: int, float*
Specifies the minimum value of the visible slider displayed when the **slider** hint is true

**slidermax** *supported types: int, float*
Specifies the maximum value of the visible slider displayed when the **slider** hint is true

**slidercenter** *supported types: int, float*
Specifies the origin value of the visible slider displayed when the **slider** hint is true

**sliderexponent** *supported types: int*
Specifies a non-linear scaling of the visible slider displayed when the **slider** hint is true

---

## "string"

This is the default widget used for string parameters. It supports these specialized hints:

**replaceRegex** *supported type: string*
A regular expression whose matching values will be replaced upon text entry. These characters are replaced by "_" unless specified by a matching **replaceWith** hint. In the example below, all non-alphanumeric and underscore characters are replaced with an underscore upon commit.

```
<param name="someParameter" replaceRegex="[^A-Za-z0-9_]"/>
```

**replaceWith** *supported type: string*
Used with **replaceRegex** to filter characters on text entry. all non-alphanumeric, underscore and space characters are replaced with an " " upon commit.

```
<param name="someParameter" replaceRegex="[^A-Za-z0-9_ ]"
replaceWith=" "/>
```

---

## "boolean"

The **boolean** widget type displays a pop-up menu with "Yes" and "No" choices. It works on either string parameters (which receive literal "Yes" and "No" values) or numeric parameters (which receive 0 and 1 values). It has no widget-specific hints.

## “checkBox”

The **checkBox** widget behaves identically to the **boolean** widget but is displayed a check box rather than a pop-up. It has no widget-specific hints.

---

## “popup”

The **popup** widget displays a pop-up menu or combo box with literal choices for parameter values. It supports these hints:

**options**: *supported types: list or pipe-delimited string*
This is the list of choices to be displayed in the pop-up menu or combo box. It may be specified as a child **hintlist** element or as a pipe-delimited string. The following two examples are equivalent.

```
<param name=“someParameter” widget=“popup” options=“a|b|c”/>
<param name=“anotherParameter” widget=“popup”>
 <hintlist name=“options”>
  <string value=“a”/>
  <string value=“b”/>
  <string value=“c”/>
 </hintlist>
</param>
```

**editable**: *supported type: boolean*
If true, the pop-up menu will instead be displayed as a combo box which support direct text entry of values not in the options list. This can also be done by using “*” as the value in the option list. The following three examples are equivalent:

```
<param name=“someParameter” widget=“popup” options=“a|b|c”
editable=“true”/>
<param name=“anotherParameter” widget=“popup” options=“a|b|c|*/>
<param name=“yetAnotherParameter” widget=“popup”>
 <hintlist name=“options”>
  <string value=“a”/>
  <string value=“b”/>
  <string value=“c”/>
  <string value=“*”/>
 </hintlist>
</param>
```

---

## “mapper”

The **mapper** widget presents a pop-up menu with associative choices. Each value has an associated display value. It supports this specific hint:

**options**: *supported types: dictionary or |-delimited string with "key:value" pairs*
The dictionary keys are the ordered display values of the pop-up menu. Each is paired with a unique literal value for the parameter. This may be specified as either a string property of a **hintdict** child element. The following options are equivalent:

```
<param name="someParameter" widget="mapper" options="a:1.0|b:2.0|c:
3.0"/>
<param name="anotherParameter" widget="mapper">
 <hintdict name="options">
  <float name="a" value="1.0"/>
  <float name="b" value="2.0"/>
  <float name="c" value="3.0"/>
 </hintdict>
</param>
```

---

## "null"

The **null** widget type is constructed but always hidden. It supports no hints.

---

## "fileInput"

The **filename** widget is a specialized string field which supports browsing for file paths via a browser dialog. It supports these specialized hints:

**typefilter**: *supported types: string*
This allows for a space-separated list of glob patterns to limit the display of files within the browser dialog.

```
<param name="someParameter" widget="filename" typefilter="*.tx *.exr"/
>
```

**presets:** *supported types: list, |-delimited string.*
When specified, a menu of preset options is available to the **filename** widget or browser dialog. These may be specified in these two forms:

```
<param name="someParameter" widget="filename" presets="red.tx|
green.tx|blue.tx"/>
<param name="anotherParameter" widget="filename">
 <hintlist name="presets">
  <string value="red.tx"/>
  <string value="green.tx"/>
  <string value="blue.tx"/>
 </hintlist>
</param>
```

**filetype**: *supported types: string*
Specifies the intended use of the file. If a value of **image** is used, the widget will provide an action to open the value in the default image viewer configured for the application.

## Conditional State Hints

Conditional Visibility and Locking allow for parameters and pages to be conditionally visible or locked (for editing) based on value comparisons of other parameters. Hints representing the parse tree of a conditional statement are supported as string hints or string entries within a dictionary hint. (Please note that Katana 2.0 supports a higher level grammar for more concisely specifying these hints. RfK does not currently take advantage of the grammar but will in a near-future release.)

When specifying hints for visibility, the root hint is named **conditionalVisOp**. For locking, it's **conditionalLockOp**. When using a **hintdict** element to store the hints, that **hintdict** itself is named **conditionalVisOps** and **conditionalLockOps** respectively. All examples will be displayed in the **hintdict** form and describe conditional visibility.

Each conditional is specified with a prefix name suffixed with "Op". The prefix for the root-level condition is always "conditionalVis". Operators which compare values address the referenced parameters via posix-style paths relative to the parameter or page on which the hint is defined. Page names are included in the parameter path. A conditional expression which evaluates to false results in the hinted widget being hidden. Comparison values are always specified as strings and converted to the type of the parameter against which they're compared.

The available operators and their operands are:

**isEqualTo**: {prefix}**Path** and {prefix}**Value**
Does an equality test with the value specified by {prefix}**Path** of the specified parameter against the provided {prefix}**Value**.

**isNotEqualTo**: {prefix}**Path** and {prefix}**Value**
Does an equality test with the value specified by {prefix}**Path** of the specified parameter against the provided {prefix}**Value**.

**isGreaterThan**: {prefix}**Path** and {prefix}**Value**
Does an comparison test with the value specified by {prefix}**Path** of the specified parameter against the provided {prefix}**Value**.

**isLessThan**: {prefix}**Path** and {prefix}**Value**
Does an comparison test with the value specified by {prefix}**Path** of the specified parameter against the provided {prefix}**Value**.

**isGreaterThanOrEqualTo**: {prefix}**Path** and {prefix}**Value**
Does an comparison test with the value specified by {prefix}**Path** of the specified parameter against the provided {prefix}**Value**.

**isLessThanOrEqualTo**: {prefix}**Path** and {prefix}**Value**
Does an comparison test with the value specified by {prefix}**Path** of the specified parameter against the provided {prefix}**Value**.

**and**: {prefix}**Left** and {prefix}**Right**

The values of the {prefix}**Left** and {prefix}**Right** operands are the prefixes of additional conditional operators. If both evaluate to true, this operator also evaluates to true.

**or**: {prefix}**Left** and {prefix}**Right**
The values of the {prefix}**Left** and {prefix}**Right** operands are the prefixes of additional conditional operators. If either evaluates to true, this operator also evaluates to true.

Here is a simple example in which we'll hint a subpage to be visible only if the parameter value of its peer is non-zero.

```
<page name="Lighting">
 <param name="someParameter"/>
 <param name="showAdvancedOptions" widget="checkBox"/>
 <page name="Advanced">
  <hintdict name="conditionalVisOps">
   <string name="conditionalVisOp" value="notEqualTo"/>
   <string name="conditionalVisPath" value="../showAdvancedOptions"/>
   <string name="conditionalVisValue" value="0"/>
  </hintdict>
  <param name="anotherParameter"/>
 </page>
</page>
```

Another example demonstrating the **or** operator and the relationship of {prefix} in the operand values. This will display the "Advanced" page if either the "showAdvancedOptions" parameter is non-zero or the "someParameter" value is greater than 10.0.

```
<page name="Lighting">
 <param name="someParameter"/>
 <param name="showAdvancedOptions" widget="checkBox"/>
 <page name="Advanced">
  <hintdict name="conditionalVisOps">
   <string name="conditionalVisOp" value="or"/>
   <string name="conditionalVisLeft" value="conditionalVis1"/>
   <string name="conditionalVisRight" value="conditionalVis2"/>
   <string name="conditionalVis1Op" value="notEqualTo"/>
   <string name="conditionalVis1Path" value="../showAdvancedOptions"/>
   <string name="conditionalVis1Value" value="0"/>
   <string name="conditionalVis2Op" value="isGreaterThan"/>
   <string name="conditionalVis2Path" value="../someParameter"/>
   <string name="conditionalVis2Value" value="1.0"/>
  </hintdict>
  <param name="anotherParameter"/>
 </page>
</page>
```

Please note that only the root operation is required to have a {prefix} starting with "conditionalVis". The prefixes of the second and third operations are defined here by the values of "conditionalVisLeft" and "conditionalVisRight" — and could be anything.

TODO: describe how to access the Conditional State Grammar parser from python in Katana to build these values.
TODO: For versions of RfK supporting Katana 2.0v1b1 and higher, add support for specifying Conditional State Grammar expressions directly in the Args files and metadata.