## File Format

Synth's file format ([dtd](http://docs.google.com/synth.dtd)) allows for specifying all the pieces necessary to create your own look and feel. A synth file is loaded by way of the [SynthLookAndFeel.load(InputStream, Class)](http://docs.google.com/javax/swing/plaf/synth/SynthLookAndFeel.html#load(java.io.InputStream,%20java.lang.Class)) or [SynthLookAndFeel.load(URL)](http://docs.google.com/javax/swing/plaf/synth/SynthLookAndFeel.html#load(java.net.URL)) methods. The following example uses the load method to configure a SynthLookAndFeel and sets it as the current look and feel:

SynthLookAndFeel laf = new SynthLookAndFeel();  
 laf.load(MyClass.class.getResourceAsStream("laf.xml"), MyClass.class);  
 UIManager.setLookAndFeel(laf);

This example loads the look and feel from an input stream, using the specified class as the resource base to resolve paths. It is also possible to load a look and feel from an arbitrary URL as in the following example.

SynthLookAndFeel laf = new SynthLookAndFeel();  
 laf.load(new URL("file:///C:/java/synth/laf/laf.xml"));  
 UIManager.setLookAndFeel(laf);

The method [SynthLookAndFeel.load(URL)](http://docs.google.com/javax/swing/plaf/synth/SynthLookAndFeel.html#load(java.net.URL)) can be used, for instance, to load a look and feel from any of the following:

* File, e.g. file:///C:/java/synth/laf/laf.xml
* Web server, e.g. http://host/laf.xml
* JAR file, e.g. jar:file:///C:/synth-laf.jar!/laf.xml
* Remote JAR file, e.g. jar:http://host/synth-laf.jar!/laf.xml

While the DTD for synth is specified, the parser is not validating. Parsing will fail only if a necessary attribute is not specified, or of the wrong type.

### The synth element

<!ELEMENT synth (([%beansPersistance;](#_1rvwp1q)) | [style](#3znysh7) | [bind](#vx1227) | [font](#3rdcrjn) | [color](#2jxsxqh) |  
 [imagePainter](#nmf14n) | [imageIcon](#1egqt2p) | [defaultsProperty](#qsh70q))\*>  
<!ATTLIST synth  
 [version](#1fob9te) CDATA #IMPLIED  
>

*Attribute definitions*

version File format version, should be 1

The [synth](#30j0zll) element contains all the other elements that make up a SynthLookAndFeel definition.

### The style element

<!ELEMENT style ([property](#2xcytpi) | [defaultsProperty](#qsh70q) | [state](#3dy6vkm) | [font](#3rdcrjn) | [graphicsUtils](#2p2csry) |  
 [insets](#3o7alnk) | [painter](#2u6wntf) | [imagePainter](#nmf14n) | [opaque](#sqyw64) | ([%beansPersistance;](#4bvk7pj)) |  
 [imageIcon](#1egqt2p))\*>  
<!ATTLIST style  
 [id](#2et92p0) ID #IMPLIED  
 [clone](#tyjcwt) IDREF #IMPLIED  
>

*Attribute definitions*

id Unique identifier for the style. clone Identifier of a previously defined style that is copied and used for the new style. This provides a conveniant mechanism for overriding only a portion of an existing style.

A style element corresponds to a SynthStyle, with the child elements specifying properties that apply to all states or state elements which contain properties specific to a particular state. The following example creates an opaque style with the id button, insets of 4, 4, 4, 4 and a font of Dialog 12.

<style id="button">  
 <opaque value="true"/>  
 <insets top="4" left="4" right="4" bottom="4"/>  
 <font name="Dialog" size="12"/>  
</style>

The following example creates a new style with an id of clonedButton that is a copy of the style with id button and has a font of Dialog, 14. The resulting style will be opaque, have insets of 4, 4, 4, 4 and a font of Dialog 14.

<style id="clonedButton" clone="button">  
 <font name="Dialog" size="14"/>  
</style>

### The state element

<!ELEMENT state ([color](#2jxsxqh) | [font](#3rdcrjn) | [painter](#2u6wntf) | [imagePainter](#nmf14n) | ([%beansPersistance;](#_1rvwp1q)) |  
 [property](#2xcytpi) | [imageIcon](#1egqt2p))\*>  
<!ATTLIST state  
 [id](#1t3h5sf) ID #IMPLIED  
 [clone](#4d34og8) IDREF #IMPLIED  
 [value](#2s8eyo1) CDATA #IMPLIED  
 [idref](#17dp8vu) IDREF #IMPLIED  
>

*Attribute definitions*

id Unique identifier for the state. clone Identifier of a previously defined state that is copied and used for the new state. value Identifies the state of the Component the properties are to apply to. This is a list of: ENABLED, MOUSE\_OVER, PRESSED, DISABLED, FOCUSED, SELECTED or DEFAULT. Multiple states should be separated by 'and.' If you do not specify a value, the contents apply to all states. idref Indicates this state should be the same as a previously defined state. This is useful for multiple styles that wish to share the same visual properties for a particular state.

The [state](#3dy6vkm) element specifies the visual properties that are to be used for a particular state of a component. For example, you could specify the background color when the Component is enabled should look different than the background color when the component is disabled. Not all Components support all states. For example, a Panel only supports the states ENABLED and DISABLED.

The following example creates a state with a red background that will be used when the component is in an a selected and pressed state:

<state value="SELECTED AND PRESSED">  
 <color value="RED" type="BACKGROUND"/>  
</state>

The state with the most individual matches will be chosen. For example, the following defines two states:

<state value="SELECTED and PRESSED" id="one">  
 <color value="RED" type="BACKGROUND"/>  
</state>  
<state value="SELECTED" id="two">  
 <color value="RED" type="BACKGROUND"/>  
</state>

State one is used when the Component is SELECTED and PRESSED, and state two when the Component is SELECTED. If the state of the Component contains at least SELECTED and PRESSED, state one will be chosen, otherwise if the state is SELECTED, but not does not contain PRESSED, state two will be used.

### The font element

<!ELEMENT font EMPTY>  
<!ATTLIST font  
 [id](#26in1rg) ID #IMPLIED  
 [clone](#lnxbz9) IDREF #IMPLIED  
 [name](#35nkun2) CDATA #IMPLIED  
 [style](#1ksv4uv) CDATA #IMPLIED  
 [size](#44sinio) CDATA #IMPLIED  
>

*Attribute definitions*

id Unique identifier for the Font. idref Identifier of a previously defined font. name Name of the font. style Style of the font. This is a list of the styles defined by Font separated by spaces: PLAIN, BOLD or ITALIC. If unspecified PLAIN is used. size Size of the font, in pixels

Defines the font for the current [state](#3dy6vkm), or [style](#3znysh7). You must specify either an [idref](#17dp8vu) or a [name](#2r0uhxc) and [size](#1664s55).

The following example creates a style with a Font of Dialog 12 Bold.

<style id="test">  
 <font name="DIALOG" size="12" style="BOLD"/>  
</style>

The following example creates a style with a font of Dialog 12 Bold that will be used if the component is ENABLED, otherwise Dialog 12 Italic will be used.

<style id="test">  
 <font name="DIALOG" size="12" style="ITALIC"/>  
 <state value="ENABLED">  
 <font name="DIALOG" size="12" style="BOLD"/>  
 </state>  
</style>

While you can supply a different font per state, in general widgets will NOT revalidate when the state changes, so that you may run into sizing problems if you try to use a font with a significantly different size for different states.

### The color element

<!ELEMENT color EMPTY>  
<!ATTLIST color  
 [id](#z337ya) ID #IMPLIED  
 [idref](#3j2qqm3) IDREF #IMPLIED  
 [type](#1y810tw) CDATA #IMPLIED  
 [value](#4i7ojhp) CDATA #IMPLIED  
>

*Attribute definitions*

id Unique identifier for the color. idref Identifier of a previously defined color. type Describes where this color should be used. This is typically one of the constants defined by ColorType: FOREGROUND, BACKGROUND, TEXT\_FOREGROUND, TEXT\_BACKGROUND or FOCUS. Alternatively you can specify the complete path to a class and field, for example javax.swing.plaf.synth.ColorType.FOREGROUND, this is useful for subclasses of synth that define additional color types. value Value for the color. This accepts the following forms.

* The name of a constant in the Color class, for example RED.
* A hex value of the form #RRGGBB where RR gives the red component, GG the green component and BB the blue component. You need not specify all color components. For example, #123 is equivalent to #000123.
* A hex value of the form #ARRGGBB or #AARRGGBB. This is useful for alpha values other than 0xFF. The form #ARRGGBB is equivalent to #0ARRGGBB.

[Color](#2jxsxqh) defines a color and what portion of the Component it should be applied to. The following example will use a background color of RED when the component is enabled.

<state value="ENABLED">  
 <color value="RED" type="BACKGROUND"/>  
</state>

The following example will have a red background when the Component is enabled, otherwise blue.

<style id="test">  
 <state value="ENABLED">  
 <color value="RED" type="BACKGROUND"/>  
 </state>  
 <state>  
 <color value="#00FF00" type="BACKGROUND"/>  
 </state>  
</style>

### The property element

<!ELEMENT property EMPTY>  
<!ATTLIST property  
 [key](#1ci93xb) CDATA #REQUIRED  
 [type](#3whwml4) (idref|boolean|dimension|insets|integer|string) "idref"  
 [value](#2bn6wsx) CDATA #REQUIRED  
>

*Attribute definitions*

key Name of the property. type Indicates the type of the property. value Value for the property. For boolean properties this will be be true or false, for integer properties this will be a valid integer, for dimensions this will be the width and height separated by a space, for insets properties this will be the top, left, bottom and right separated by a space and for idref properties this will be the unique id of a previously defined object.

[Property](#2xcytpi) elements are used to add key value pairs to a SynthStyle that can be accessed by way of the get method. Many Components use the key value pairs for configuring their visual appearance. Refer to [property table](http://docs.google.com/componentProperties.html) for a list of the properties each Component supports. The following creates the properties ScrollBar.allowsAbsolutePositioning, OptionPane.minimumSize, ScrollPane.viewportBorderInsets, Tree.rowHeight and foreground with the values false, a dimensions of 262x90, an insets of 5, 5, 5, 5, the integer 20 and an instance of the class ArrowButtonPainter.

<style id="test">  
 <property key="ScrollBar.allowsAbsolutePositioning" type="boolean" value="false"/>  
 <property key="OptionPane.minimumSize" type="dimension" value="262 90"/>  
 <property key="ScrollPane.viewportBorderInsets" type="insets" value="5 5 5 5"/>  
 <property key="Tree.rowHeight" type="integer" value="20"/>  
 <object class="ArrowButtonPainter" id="ArrowButtonPainter"/>  
 <property key="foreground" type="idref" value="ArrowButtonPainter"/>  
</style>

You can also specify properties that are to apply to specific states. Whether or not the property is accessed for each state depends upon how the property is used. For example, the following specifies a default icon and an icon to use while the mouse is over the component.

<style id="test">  
 <imageIcon id="defaultIcon" path="resources/myImage.png"/>  
 <property key="RadioButton.icon" value="defaultIcon"/>  
 <state value="MOUSE\_OVER">  
 <imageIcon id="mouseOverIcon" path="resources/myMouseOverImage.png"/>  
 <property key="RadioButton.icon" value="mouseOverIcon"/>  
 </state>  
</style>

<!ELEMENT defaultsProperty EMPTY>  
<!ATTLIST defaultsProperty  
 [key](#3as4poj) CDATA #REQUIRED  
 [type](#1pxezwc) (idref|boolean|dimension|insets|integer|string) "idref"  
 [value](#49x2ik5) CDATA #REQUIRED  
>

*Attribute definitions*

key Name of the property. type Indicates the type of the property. value Value for the property. For boolean properties this will be true or false, for integer properties this will be a valid integer, for dimensions this will be the width and height separated by a space, for insets properties this will be the top, left, bottom and right separated by a space and for idref properties this will be the unique id of a previously defined object.

[DefaultsProperty](#qsh70q) elements are used to define properties that will be placed in the UIDefaults table that SynthLookAndFeel supplies to the UIManager. The following assigns the the Color red to the value Table.focusCellForeground.

<style id="test">  
 <object class="javax.swing.plaf.ColorUIResource" id="color">  
 <int>255</int>  
 <int>0</int>  
 <int>0</int>  
 </object>  
 <defaultsProperty key="Table.focusCellForeground" type="idref" value="color"/>  
</style>

This value could then be asked by way of UIManager.get("Table.focusCellForeground").

### The graphicsUtils element

<!ELEMENT graphicsUtils EMPTY>  
<!ATTLIST graphicsUtils  
 [idref](#147n2zr) IDREF #REQUIRED  
>

*Attribute definitions*

idref Identifer of a previously defined SynthGraphicsUtils object that is to be used as the SynthGraphicsUtils for the current [style](#3znysh7).

[GraphicsUtils](#2p2csry) elements are used to define the SynthGraphicsUtils that the current [style](#3znysh7) will use. The following example creates a style with an instance of CustomGraphicsUtils for the SynthGraphicsUtils.

<style id="test">  
 <object class="CustomGraphicsUtils" id="graphics"/>  
 <graphicsUtils idref="graphics"/>  
</style>

### The insets element

<!ELEMENT insets EMPTY>  
<!ATTLIST insets  
 [id](#23ckvvd) ID #IMPLIED  
 [idref](#ihv636) IDREF #IMPLIED  
 [top](#32hioqz) CDATA #IMPLIED  
 [bottom](#1hmsyys) CDATA #IMPLIED  
 [left](#41mghml) CDATA #IMPLIED  
 [right](#2grqrue) CDATA #IMPLIED  
>

*Attribute definitions*

id Unique identifier for the Insets. idref Identifier of a previously defined Insets. top Top component of the Insets. bottom Bottom component of the Insets. left Left component of the Insets. right Right component of the Insets.

[Insets](#3o7alnk) elements are used to define the Insets for the current [style](#3znysh7). The insets will be set on any Components the [style](#3znysh7) is associated with. The following example creates a style with insets of 1, 2, 3, 0.

<style id="test">  
 <insets top="1" bottom="2" left="3"/>  
</style>

### The bind element

<!ELEMENT bind EMPTY>  
<!ATTLIST bind  
 [style](#3fwokq0) IDREF #REQUIRED  
 [type](#1v1yuxt) (name|region) #REQUIRED  
 [key](#4f1mdlm) CDATA #REQUIRED  
>

*Attribute definitions*

style Unique identifier of a previously defined style. type One of name or region. For type name component.getName() is used, otherwise the name of the Region is used. key Regular expression applied to the name of the Component, or the name of the Region, depending upon the value of [type](#1v1yuxt).

[Bind](#vx1227) elements specify which Regions a style is to be used for. The following example applies the [style](#3znysh7) test to any Component whose name starts with test.

<style id="test">  
 <insets top="1" bottom="2" left="3"/>  
</style>  
<bind style="test" type="name" key="test.\*"/>

Numerous styles may apply to a region, in which case each of the matching styles is merged into a resulting style that is used. Precedence is given to styles defined later in the file. For example, the following defines two styles, a and b. Style a is applied to any component with a name starting with test, and style b is used for button regions.

<style id="a">  
 <font name="DIALOG" size="12" style="ITALIC"/>  
 <insets top="1" bottom="2" left="3"/>  
</style>  
<bind style="a" type="name" key="test.\*"/>  
<style id="b">  
 <font name="DIALOG" size="12" style="BOLD"/>  
</style>  
<bind style="b" type="region" key="button"/>

For a button with the name test this is equivalent to:

<style>  
 <font name="DIALOG" size="12" style="BOLD"/>  
 <insets top="1" bottom="2" left="3"/>  
</style>

Merging happens for states of a style as well.

<style id="a">  
 <font name="DIALOG" size="12" style="ITALIC"/>  
 <insets top="1" bottom="2" left="3"/>  
 <state value="ENABLED">  
 <object id="customPainter" class="CustomPainter"/>  
 <painter idref="customPainter"/>  
 </state>  
</style>  
<bind style="a" type="name" key="test.\*"/>  
<style id="b">  
 <font name="DIALOG" size="12" style="BOLD"/>  
 <state value="ENABLED">  
 <font name="Lucida" size="12" style="ITALIC"/>  
 </state>  
</style>  
<bind style="b" type="region" key="button"/>

For a button with the name test this is equivalent to:

<style>  
 <font name="DIALOG" size="12" style="BOLD"/>  
 <insets top="1" bottom="2" left="3"/>  
 <state value="ENABLED">  
 <object id="customPainter" class="CustomPainter"/>  
 <painter idref="customPainter"/>  
 <font name="Lucida" size="12" style="ITALIC"/>  
 </state>  
</style>

### The painter element

<!ELEMENT painter EMPTY>  
<!ATTLIST painter  
 [idref](#19c6y18) IDREF #IMPLIED  
 [method](#3tbugp1) CDATA #IMPLIED  
 [direction](#28h4qwu) (north|south|east|west|top|left|bottom|right|horizontal|vertical|horizontal\_split|vertical\_split) #IMPLIED  
>

*Attribute definitions*

idref Identifier of a previously defined SynthPainter. method Identifies the SynthPainter method this is to be used for. The name corresponds to the method name of a paint method in SynthPainter with the paint prefix dropped, the remainder is case insensitive (using the latin1 case folding rules). For example SynthPainter.paintButtonBackground is identified by 'buttonBackground' or 'buttonbackground'. If this is not specified the painter is used for all methods that don't have a a specific painter for them. direction Identifies the direction, or orientation, this painter is to be used for. This is only useful for the SynthPainter methods that take a direction or orientation. If this is not specified the painter is used for all directions.

[Painter](#2u6wntf) defines a SynthPainter for the current style or the state of the current style. The following example binds an instance of the class MyPainter which must be a SynthPainter to the style test.

<style id="test">  
 <object class="MyPainter" id="MyPainter"/>  
 <painter idref="MyPainter"/>  
</style>

The painter that is used for a particular method and state is determined as follows:

1. Painter specified for the current state, method and direction.
2. Painter specified for the current state and method.
3. Painter specified for the current state.
4. Painter specified for the style, method and direction.
5. Painter specified for the style and method.
6. Painter specified for the style.

Consider the following:

<style id="test">  
 <painter idref="fallbackPainter"/>  
 <painter idref="styleButtonBackgroundPainter" method="buttonBackground"/>  
 <state value="SELECTED">  
 <painter idref="stateFallbackPainter"/>  
 <painter idref="stateButtonBackgroundPainter" method="buttonBackground"/>  
 </state>  
</style>

The following outlines which painter will be used for what SynthPainter method:

| State | Method | Painter |
| --- | --- | --- |
| SELECTED | paintButtonBackground | stateButtonBackgroundPainter |
| SELECTED | Anything but paintButtonBackground | stateFallbackPainter |
| Anything but SELECTED | paintButtonBackground | styleButtonBackgroundPainter |
| Anything but SELECTED | Anything but paintButtonBackground | fallbackPainter |

When several identical painters are declared, they are aggregated into a single one. Two painters are identical if their *direction* and *method* attributes values are equal. Consider the following:

<style id="panelStyle">  
 <imagePainter method="panelBackground" path="red.png" />  
 <imagePainter method="panelBackground" path="green.png" />  
 <imagePainter method="panelBackground" path="blue.png" />  
</style>

These three painters are identical for they use the same method and the same direction (all directions by default). Synth aggregates these painters to create a single one that will paint its children painters in the order of declaration. Hence, Synth will first paint the red picture, then the green one and finally the blue one. Each child painter can be seen as a layer of the aggregate painter.

Painter aggregation, or multi-layering, is very useful to reuse elements. Imagine you want to use an highlight effect on buttons and on selected menu items. With painter aggregation, you just need to create a separate highlighting painter instead of having buttons and menu items painters handle it.

### The imagePainter element

<!ELEMENT imagePainter EMPTY>  
<!ATTLIST imagePainter  
 [id](#37m2jsg) ID #IMPLIED  
 [method](#1mrcu09) CDATA #IMPLIED  
 [direction](#46r0co2) (north|south|east|west|top|left|bottom|right|horizontal|vertical|horizontal\_split|vertical\_split) #IMPLIED  
 [path](#2lwamvv) CDATA #REQUIRED  
 [sourceInsets](#111kx3o) CDATA #IMPLIED  
 [destinationInsets](#3l18frh) CDATA #IMPLIED  
 [paintCenter](#3q5sasy) (true|false) "true"  
 [stretch](#4k668n3) (true|false) "true"  
 [center](#2zbgiuw) (true|false) "false"  
>

*Attribute definitions*

id Unique identifier for the imagePainter. method Identifies the SynthPainter method this is to be used for. The name corresponds to the method name of a paint method in SynthPainter with the paint prefix dropped, the remainder is case insensitive (using the latin1 case folding rules). For example SynthPainter.paintButtonBackground is identified by 'buttonBackground' or 'buttonbackground'. If this is not specified the painter is used for all methods that don't have a a specific painter for them. direction Identifies the direction, or orientation, this image is to be used for. This is only useful for the SynthPainter methods that take a direction or orientation. If this is not specified the image is used for all directions. path Path to the image. Path to the image. If SynthLookAndFeel.load is passed a Class this will use the Class method getResource (with with the Class suplied to the load method). If load is passed a URL this will use the URL constructor URL(context, path) to resolve the path. sourceInsets Insets on the source image. This is top, left, bottom, right with each component separated by a space. destinationInsets Insets of the destination image. This is top, left, bottom, right with each component separated by a space. If not specified the [sourceInsets](#111kx3o) are used. paintCenter Whether or not the center of the image should be drawn. stretch Whether or not the north, south, east and west components of the resulting image should be scaled or tiled. center Whether or not the image is centered.

The [ImagePainter](#nmf14n) element defines a painter for the current style or state that will render using the specified image. ImagePainter offers two distinct rendering modes. The first mode is used to center an image in the space provided. This is commonly used in rendering decorations on top of a widget, for example, to specify an arrow for a scroll button use the center mode. The following example illustrates this:

<style id="test">  
 <imagePainter path="resources/myImage.png" center="true"/>  
</style>

The second mode is used in scaling an image to fit in the provided space. In this mode sourceInsets is used to specify a border around an image where the north, south, east and west edges of the border are either stretched or tiled (stretch attribute), the four corners of the border drawn in place, and the center is stretched. In this mode you must specify sourceInsets. The following example illustrates an ImagePainter that is using the image MyImage.png and insets of 2 all the way around:

<style id="test">  
 <imagePainter path="resources/myImage.png"  
 sourceInsets="2 2 2 2"/>  
</style>

Refer to the description of the [painter](#2u6wntf) element for details as to the precedence in choosing a painter and to understand how identical painters are handled.

### The imageIcon element

<!ELEMENT imageIcon EMPTY>  
<!ATTLIST imageIcon  
 [id](#3ygebqi) ID #REQUIRED  
 [path](#2dlolyb) CDATA #REQUIRED  
>

*Attribute definitions*

id Unique identifier for the imageIcon. path Path to the image. This uses the Class method getResource to resolve the path, with the Class supplied to SynthLookAndFeel.load.

[ImageIcon](#1egqt2p) is used to assign an Icon implementation that is wrapping an Image to a unique identifier. This is typically used for properties that take an Icon. The following example binds an ImageIcon to the property RadioButton.icon.

<style id="test">  
 <imageIcon id="icon" path="resources/myImage.png"/>  
 <property key="RadioButton.icon" value="icon"/>  
</style>

### The opaque element

<!ELEMENT opaque EMPTY>  
<!ATTLIST opaque  
 [value](#3cqmetx) (true|false) "true"  
>

*Attribute definitions*

id Whether or not the style should be opaque, if unspecified the style is opaque.

The [opaque](#sqyw64) element indicates whether or not any Components the style is associated with are to be made opaque. The painter will be asked to paint regardless of the opacity of the associated Component. The following example creates a style that is not opaque.

<style id="test">  
 <opaque value="FALSE">  
 <painter idref="painter"/>  
</style>

### The beansPersistance entity

Beans persistance can be used to embed any Object. This is typically used for embedding your own Painters, but can be used for other arbritrary objects as well. Refer to <http://java.sun.com/products/jfc/tsc/articles/persistence3/> for details on beans persistance.

### Backing Style

In creating a Synth file it's good practice to create a backing style that is used by all components. This will make sure that any components that do not match a specific style will have a default font, foreground, background and opacity. The following example illustrates this:

<synth>  
 <style id="backingStyle">  
 <opaque value="true"/>  
 <font name="Dialog" size="12"/>  
 <state>  
 <color value="BLACK" type="BACKGROUND"/>  
 <color value="WHITE" type="FOREGROUND"/>  
 </state>  
 </style>  
 <bind style="backingStyle" type="region" key=".\*"/>  
</synth>