| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/LookAndFeel.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/ListSelectionModel.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/MenuElement.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/LookAndFeel.html)    [**NO FRAMES**](http://docs.google.com/LookAndFeel.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: FIELD | [CONSTR](#4d34og8) | [METHOD](#17dp8vu) |

## **javax.swing**

Class LookAndFeel

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **javax.swing.LookAndFeel**

**Direct Known Subclasses:** [BasicLookAndFeel](http://docs.google.com/javax/swing/plaf/basic/BasicLookAndFeel.html), [MultiLookAndFeel](http://docs.google.com/javax/swing/plaf/multi/MultiLookAndFeel.html)

public abstract class **LookAndFeel**extends [Object](http://docs.google.com/java/lang/Object.html)

LookAndFeel, as the name implies, encapsulates a look and feel. Beyond installing a look and feel most developers never need to interact directly with LookAndFeel. In general only developers creating a custom look and feel need to concern themselves with this class.

Swing is built upon the foundation that each JComponent subclass has an implementation of a specific ComponentUI subclass. The ComponentUI is often referred to as "the ui", "component ui", or "look and feel delegate". The ComponentUI subclass is responsible for providing the look and feel specific functionality of the component. For example, JTree requires an implementation of the ComponentUI subclass TreeUI. The implementation of the specific ComponentUI subclass is provided by the LookAndFeel. Each JComponent subclass identifies the ComponentUI subclass it requires by way of the JComponent method getUIClassID.

Each LookAndFeel implementation must provide an implementation of the appropriate ComponentUI subclass by specifying a value for each of Swing's ui class ids in the UIDefaults object returned from getDefaults. For example, BasicLookAndFeel uses BasicTreeUI as the concrete implementation for TreeUI. This is accomplished by BasicLookAndFeel providing the key-value pair "TreeUI"-"javax.swing.plaf.basic.BasicTreeUI", in the UIDefaults returned from getDefaults. Refer to [UIDefaults.getUI(JComponent)](http://docs.google.com/javax/swing/UIDefaults.html#getUI(javax.swing.JComponent)) for defails on how the implementation of the ComponentUI subclass is obtained.

When a LookAndFeel is installed the UIManager does not check that an entry exists for all ui class ids. As such, random exceptions will occur if the current look and feel has not provided a value for a particular ui class id and an instance of the JComponent subclass is created.

## Recommendations for Look and Feels

As noted in UIManager each LookAndFeel has the opportunity to provide a set of defaults that are layered in with developer and system defaults. Some of Swing's components require the look and feel to provide a specific set of defaults. These are documented in the classes that require the specific default.

### ComponentUIs and defaults

All ComponentUIs typically need to set various properties on the JComponent the ComponentUI is providing the look and feel for. This is typically done when the ComponentUI is installed on the JComponent. Setting a property should only be done if the developer has not set the property. For non-primitive values it is recommended that the ComponentUI only change the property on the JComponent if the current value is null or implements UIResource. If the current value is null or implements UIResource it indicates the property has not been set by the developer, and the ui is free to change it. For example, BasicButtonUI.installDefaults only changes the font on the JButton if the return value from button.getFont() is null or implements UIResource. On the other hand if button.getFont() returned a non-null value that did not implement UIResource then BasicButtonUI.installDefaults would not change the JButton's font.

For primitive values, such as opaque, the method installProperty should be invoked. installProperty only changes the correspoding property if the value has not been changed by the developer.

ComponentUI implementations should use the various install methods provided by this class as they handle the necessary checking and install the property using the recommended guidelines.

### Exceptions

All of the install methods provided by LookAndFeel need to access the defaults if the value of the property being changed is null or a UIResource. For example, installing the font does the following:

JComponent c;  
 Font font = c.getFont();  
 if (font == null || (font instanceof UIResource)) {  
 c.setFont(UIManager.getFont("fontKey"));  
 }

If the font is null or a UIResource, the defaults table is queried with the key fontKey. All of UIDefault's get methods throw a NullPointerException if passed in null. As such, unless otherwise noted each of the various install methods of LookAndFeel throw a NullPointerException if the current value is null or a UIResource and the supplied defaults key is null. In addition, unless otherwise specified all of the install methods throw a NullPointerException if a null component is passed in.

| **Constructor Summary** | |
| --- | --- |
| [**LookAndFeel**](http://docs.google.com/javax/swing/LookAndFeel.html#LookAndFeel())() |

| **Method Summary** | |
| --- | --- |
| [UIDefaults](http://docs.google.com/javax/swing/UIDefaults.html) | [**getDefaults**](http://docs.google.com/javax/swing/LookAndFeel.html#getDefaults())()            Returns the look and feel defaults. |
| abstract  [String](http://docs.google.com/java/lang/String.html) | [**getDescription**](http://docs.google.com/javax/swing/LookAndFeel.html#getDescription())()            Return a one line description of this look and feel implementation, e.g. |
| static [Object](http://docs.google.com/java/lang/Object.html) | [**getDesktopPropertyValue**](http://docs.google.com/javax/swing/LookAndFeel.html#getDesktopPropertyValue(java.lang.String,%20java.lang.Object))([String](http://docs.google.com/java/lang/String.html) systemPropertyName, [Object](http://docs.google.com/java/lang/Object.html) fallbackValue)            Returns the value of the specified system desktop property by invoking Toolkit.getDefaultToolkit().getDesktopProperty(). |
| [Icon](http://docs.google.com/javax/swing/Icon.html) | [**getDisabledIcon**](http://docs.google.com/javax/swing/LookAndFeel.html#getDisabledIcon(javax.swing.JComponent,%20javax.swing.Icon))([JComponent](http://docs.google.com/javax/swing/JComponent.html) component, [Icon](http://docs.google.com/javax/swing/Icon.html) icon)            Returns an Icon with a disabled appearance. |
| [Icon](http://docs.google.com/javax/swing/Icon.html) | [**getDisabledSelectedIcon**](http://docs.google.com/javax/swing/LookAndFeel.html#getDisabledSelectedIcon(javax.swing.JComponent,%20javax.swing.Icon))([JComponent](http://docs.google.com/javax/swing/JComponent.html) component, [Icon](http://docs.google.com/javax/swing/Icon.html) icon)            Returns an Icon for use by disabled components that are also selected. |
| abstract  [String](http://docs.google.com/java/lang/String.html) | [**getID**](http://docs.google.com/javax/swing/LookAndFeel.html#getID())()            Return a string that identifies this look and feel. |
| [LayoutStyle](http://docs.google.com/javax/swing/LayoutStyle.html) | [**getLayoutStyle**](http://docs.google.com/javax/swing/LookAndFeel.html#getLayoutStyle())()            Returns the LayoutStyle for this look and feel. |
| abstract  [String](http://docs.google.com/java/lang/String.html) | [**getName**](http://docs.google.com/javax/swing/LookAndFeel.html#getName())()            Return a short string that identifies this look and feel, e.g. |
| boolean | [**getSupportsWindowDecorations**](http://docs.google.com/javax/swing/LookAndFeel.html#getSupportsWindowDecorations())()            Returns true if the LookAndFeel returned RootPaneUI instances support providing Window decorations in a JRootPane. |
| void | [**initialize**](http://docs.google.com/javax/swing/LookAndFeel.html#initialize())()            Initializes the look and feel. |
| static void | [**installBorder**](http://docs.google.com/javax/swing/LookAndFeel.html#installBorder(javax.swing.JComponent,%20java.lang.String))([JComponent](http://docs.google.com/javax/swing/JComponent.html) c, [String](http://docs.google.com/java/lang/String.html) defaultBorderName)            Convenience method for setting a component's border property with a value from the defaults. |
| static void | [**installColors**](http://docs.google.com/javax/swing/LookAndFeel.html#installColors(javax.swing.JComponent,%20java.lang.String,%20java.lang.String))([JComponent](http://docs.google.com/javax/swing/JComponent.html) c, [String](http://docs.google.com/java/lang/String.html) defaultBgName, [String](http://docs.google.com/java/lang/String.html) defaultFgName)            Convenience method for setting a component's foreground and background color properties with values from the defaults. |
| static void | [**installColorsAndFont**](http://docs.google.com/javax/swing/LookAndFeel.html#installColorsAndFont(javax.swing.JComponent,%20java.lang.String,%20java.lang.String,%20java.lang.String))([JComponent](http://docs.google.com/javax/swing/JComponent.html) c, [String](http://docs.google.com/java/lang/String.html) defaultBgName, [String](http://docs.google.com/java/lang/String.html) defaultFgName, [String](http://docs.google.com/java/lang/String.html) defaultFontName)            Convenience method for setting a component's foreground, background and font properties with values from the defaults. |
| static void | [**installProperty**](http://docs.google.com/javax/swing/LookAndFeel.html#installProperty(javax.swing.JComponent,%20java.lang.String,%20java.lang.Object))([JComponent](http://docs.google.com/javax/swing/JComponent.html) c, [String](http://docs.google.com/java/lang/String.html) propertyName, [Object](http://docs.google.com/java/lang/Object.html) propertyValue)            Convenience method for installing a property with the specified name and value on a component if that property has not already been set by the developer. |
| abstract  boolean | [**isNativeLookAndFeel**](http://docs.google.com/javax/swing/LookAndFeel.html#isNativeLookAndFeel())()            If the underlying platform has a "native" look and feel, and this is an implementation of it, return true. |
| abstract  boolean | [**isSupportedLookAndFeel**](http://docs.google.com/javax/swing/LookAndFeel.html#isSupportedLookAndFeel())()            Return true if the underlying platform supports and or permits this look and feel. |
| static void | [**loadKeyBindings**](http://docs.google.com/javax/swing/LookAndFeel.html#loadKeyBindings(javax.swing.InputMap,%20java.lang.Object%5B%5D))([InputMap](http://docs.google.com/javax/swing/InputMap.html) retMap, [Object](http://docs.google.com/java/lang/Object.html)[] keys)            Populates an InputMap with the specified bindings. |
| static [ComponentInputMap](http://docs.google.com/javax/swing/ComponentInputMap.html) | [**makeComponentInputMap**](http://docs.google.com/javax/swing/LookAndFeel.html#makeComponentInputMap(javax.swing.JComponent,%20java.lang.Object%5B%5D))([JComponent](http://docs.google.com/javax/swing/JComponent.html) c, [Object](http://docs.google.com/java/lang/Object.html)[] keys)            Creates a ComponentInputMapUIResource from keys. |
| static [Object](http://docs.google.com/java/lang/Object.html) | [**makeIcon**](http://docs.google.com/javax/swing/LookAndFeel.html#makeIcon(java.lang.Class,%20java.lang.String))([Class](http://docs.google.com/java/lang/Class.html)<?> baseClass, [String](http://docs.google.com/java/lang/String.html) gifFile)            Creates and returns a UIDefault.LazyValue that loads an image. |
| static [InputMap](http://docs.google.com/javax/swing/InputMap.html) | [**makeInputMap**](http://docs.google.com/javax/swing/LookAndFeel.html#makeInputMap(java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] keys)            Creates a InputMapUIResource from keys. |
| static [JTextComponent.KeyBinding](http://docs.google.com/javax/swing/text/JTextComponent.KeyBinding.html)[] | [**makeKeyBindings**](http://docs.google.com/javax/swing/LookAndFeel.html#makeKeyBindings(java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html)[] keyBindingList)            Convenience method for building an array of KeyBindings. |
| void | [**provideErrorFeedback**](http://docs.google.com/javax/swing/LookAndFeel.html#provideErrorFeedback(java.awt.Component))([Component](http://docs.google.com/java/awt/Component.html) component)            Invoked when the user attempts an invalid operation, such as pasting into an uneditable JTextField that has focus. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/javax/swing/LookAndFeel.html#toString())()            Returns a string that displays and identifies this object's properties. |
| void | [**uninitialize**](http://docs.google.com/javax/swing/LookAndFeel.html#uninitialize())()            Uninitializes the look and feel. |
| static void | [**uninstallBorder**](http://docs.google.com/javax/swing/LookAndFeel.html#uninstallBorder(javax.swing.JComponent))([JComponent](http://docs.google.com/javax/swing/JComponent.html) c)            Convenience method for uninstalling a border. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Constructor Detail** |
| --- |

### LookAndFeel

public **LookAndFeel**()

| **Method Detail** |
| --- |

### installColors

public static void **installColors**([JComponent](http://docs.google.com/javax/swing/JComponent.html) c,  
 [String](http://docs.google.com/java/lang/String.html) defaultBgName,  
 [String](http://docs.google.com/java/lang/String.html) defaultFgName)

Convenience method for setting a component's foreground and background color properties with values from the defaults. The properties are only set if the current value is either null or a UIResource.

**Parameters:**c - component to set the colors ondefaultBgName - key for the backgrounddefaultFgName - key for the foreground **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - as described in [exceptions](#_2et92p0)**See Also:**[installColorsAndFont(javax.swing.JComponent, java.lang.String, java.lang.String, java.lang.String)](http://docs.google.com/javax/swing/LookAndFeel.html#installColorsAndFont(javax.swing.JComponent,%20java.lang.String,%20java.lang.String,%20java.lang.String)), [UIManager.getColor(java.lang.Object)](http://docs.google.com/javax/swing/UIManager.html#getColor(java.lang.Object))

### installColorsAndFont

public static void **installColorsAndFont**([JComponent](http://docs.google.com/javax/swing/JComponent.html) c,  
 [String](http://docs.google.com/java/lang/String.html) defaultBgName,  
 [String](http://docs.google.com/java/lang/String.html) defaultFgName,  
 [String](http://docs.google.com/java/lang/String.html) defaultFontName)

Convenience method for setting a component's foreground, background and font properties with values from the defaults. The properties are only set if the current value is either null or a UIResource.

**Parameters:**c - component set to the colors and font ondefaultBgName - key for the backgrounddefaultFgName - key for the foregrounddefaultFontName - key for the font **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - as described in [exceptions](#_2et92p0)**See Also:**[installColors(javax.swing.JComponent, java.lang.String, java.lang.String)](http://docs.google.com/javax/swing/LookAndFeel.html#installColors(javax.swing.JComponent,%20java.lang.String,%20java.lang.String)), [UIManager.getColor(java.lang.Object)](http://docs.google.com/javax/swing/UIManager.html#getColor(java.lang.Object)), [UIManager.getFont(java.lang.Object)](http://docs.google.com/javax/swing/UIManager.html#getFont(java.lang.Object))

### installBorder

public static void **installBorder**([JComponent](http://docs.google.com/javax/swing/JComponent.html) c,  
 [String](http://docs.google.com/java/lang/String.html) defaultBorderName)

Convenience method for setting a component's border property with a value from the defaults. The border is only set if the border is null or an instance of UIResource.

**Parameters:**c - component to set the border ondefaultBorderName - key specifying the border **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - as described in [exceptions](#_2et92p0)

### uninstallBorder

public static void **uninstallBorder**([JComponent](http://docs.google.com/javax/swing/JComponent.html) c)

Convenience method for uninstalling a border. If the border of the component is a UIResource, it is set to null.

**Parameters:**c - component to uninstall the border on **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if c is null

### installProperty

public static void **installProperty**([JComponent](http://docs.google.com/javax/swing/JComponent.html) c,  
 [String](http://docs.google.com/java/lang/String.html) propertyName,  
 [Object](http://docs.google.com/java/lang/Object.html) propertyValue)

Convenience method for installing a property with the specified name and value on a component if that property has not already been set by the developer. This method is intended to be used by ui delegate instances that need to specify a default value for a property of primitive type (boolean, int, ..), but do not wish to override a value set by the client. Since primitive property values cannot be wrapped with the UIResource marker, this method uses private state to determine whether the property has been set by the client.

**Parameters:**c - target component to set the property onpropertyName - name of the property to setpropertyValue - value of the property **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the specified property is not one which can be set using this method [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the property value has not been set by the developer and the type does not match the property's type [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if c is null, or the named property has not been set by the developer and propertyValue is null**Since:** 1.5

### makeKeyBindings

public static [JTextComponent.KeyBinding](http://docs.google.com/javax/swing/text/JTextComponent.KeyBinding.html)[] **makeKeyBindings**([Object](http://docs.google.com/java/lang/Object.html)[] keyBindingList)

Convenience method for building an array of KeyBindings. While this method is not deprecated, developers should instead use ActionMap and InputMap for supplying key bindings.

This method returns an array of KeyBindings, one for each alternating key-action pair in keyBindingList. A key can either be a String in the format specified by the KeyStroke.getKeyStroke method, or a KeyStroke. The action part of the pair is a String that corresponds to the name of the Action.

The following example illustrates creating a KeyBinding array from six alternating key-action pairs:

JTextComponent.KeyBinding[] multilineBindings = makeKeyBindings( new Object[] {  
 "UP", DefaultEditorKit.upAction,  
 "DOWN", DefaultEditorKit.downAction,  
 "PAGE\_UP", DefaultEditorKit.pageUpAction,  
 "PAGE\_DOWN", DefaultEditorKit.pageDownAction,  
 "ENTER", DefaultEditorKit.insertBreakAction,  
 "TAB", DefaultEditorKit.insertTabAction  
 });

If keyBindingList's length is odd, the last element is ignored.

Supplying a null value for either the key or action part of the key-action pair results in creating a KeyBinding with the corresponding value null. As other parts of Swing's expect non-null values in a KeyBinding, you should avoid supplying null as either the key or action part of the key-action pair.

**Parameters:**keyBindingList - an array of key-action pairs **Returns:**an array of KeyBindings **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if keyBindingList is null [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the key part of the pair is not a KeyStroke or String, or the action part of the pair is not a String**See Also:**[ActionMap](http://docs.google.com/javax/swing/ActionMap.html), [InputMap](http://docs.google.com/javax/swing/InputMap.html), [KeyStroke.getKeyStroke(char)](http://docs.google.com/javax/swing/KeyStroke.html#getKeyStroke(char))

### makeInputMap

public static [InputMap](http://docs.google.com/javax/swing/InputMap.html) **makeInputMap**([Object](http://docs.google.com/java/lang/Object.html)[] keys)

Creates a InputMapUIResource from keys. This is a convenience method for creating a new InputMapUIResource, invoking loadKeyBindings(map, keys), and returning the InputMapUIResource.

**Parameters:**keys - alternating pairs of keystroke-action key pairs as described in [loadKeyBindings(javax.swing.InputMap, java.lang.Object[])](http://docs.google.com/javax/swing/LookAndFeel.html#loadKeyBindings(javax.swing.InputMap,%20java.lang.Object%5B%5D)) **Returns:**newly created and populated InputMapUIResource**Since:** 1.3 **See Also:**[loadKeyBindings(javax.swing.InputMap, java.lang.Object[])](http://docs.google.com/javax/swing/LookAndFeel.html#loadKeyBindings(javax.swing.InputMap,%20java.lang.Object%5B%5D))

### makeComponentInputMap

public static [ComponentInputMap](http://docs.google.com/javax/swing/ComponentInputMap.html) **makeComponentInputMap**([JComponent](http://docs.google.com/javax/swing/JComponent.html) c,  
 [Object](http://docs.google.com/java/lang/Object.html)[] keys)

Creates a ComponentInputMapUIResource from keys. This is a convenience method for creating a new ComponentInputMapUIResource, invoking loadKeyBindings(map, keys), and returning the ComponentInputMapUIResource.

**Parameters:**c - component to create the ComponentInputMapUIResource withkeys - alternating pairs of keystroke-action key pairs as described in [loadKeyBindings(javax.swing.InputMap, java.lang.Object[])](http://docs.google.com/javax/swing/LookAndFeel.html#loadKeyBindings(javax.swing.InputMap,%20java.lang.Object%5B%5D)) **Returns:**newly created and populated InputMapUIResource **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if c is null**Since:** 1.3 **See Also:**[loadKeyBindings(javax.swing.InputMap, java.lang.Object[])](http://docs.google.com/javax/swing/LookAndFeel.html#loadKeyBindings(javax.swing.InputMap,%20java.lang.Object%5B%5D)), [ComponentInputMapUIResource](http://docs.google.com/javax/swing/plaf/ComponentInputMapUIResource.html)

### loadKeyBindings

public static void **loadKeyBindings**([InputMap](http://docs.google.com/javax/swing/InputMap.html) retMap,  
 [Object](http://docs.google.com/java/lang/Object.html)[] keys)

Populates an InputMap with the specified bindings. The bindings are supplied as a list of alternating keystroke-action key pairs. The keystroke is either an instance of KeyStroke, or a String that identifies the KeyStroke for the binding. Refer to KeyStroke.getKeyStroke(String) for the specific format. The action key part of the pair is the key registered in the InputMap for the KeyStroke.

The following illustrates loading an InputMap with two key-action pairs:

LookAndFeel.loadKeyBindings(inputMap, new Object[] {  
 "control X", "cut",  
 "control V", "paste"  
 });

Supplying a null list of bindings (keys) does not change retMap in any way.

Specifying a null action key results in removing the keystroke's entry from the InputMap. A null keystroke is ignored.

**Parameters:**retMap - InputMap to add the key-action pairs tokeys - bindings to add to retMap **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if keys is non-null, not empty, and retMap is null**Since:** 1.3 **See Also:**[KeyStroke.getKeyStroke(String)](http://docs.google.com/javax/swing/KeyStroke.html#getKeyStroke(java.lang.String)), [InputMap](http://docs.google.com/javax/swing/InputMap.html)

### makeIcon

public static [Object](http://docs.google.com/java/lang/Object.html) **makeIcon**([Class](http://docs.google.com/java/lang/Class.html)<?> baseClass,  
 [String](http://docs.google.com/java/lang/String.html) gifFile)

Creates and returns a UIDefault.LazyValue that loads an image. The returned value is an implementation of UIDefaults.LazyValue. When createValue is invoked on the returned object, the image is loaded. If the image is non-null, it is then wrapped in an Icon that implements UIResource. The image is loaded using Class.getResourceAsStream(gifFile).

This method does not check the arguments in any way. It is strongly recommended that non-null values are supplied else exceptions may occur when createValue is invoked on the returned object.

**Parameters:**baseClass - Class used to load the resourcegifFile - path to the image to load **Returns:**a UIDefaults.LazyValue; when resolved the LazyValue loads the specified image**See Also:**[UIDefaults.LazyValue](http://docs.google.com/javax/swing/UIDefaults.LazyValue.html), [Icon](http://docs.google.com/javax/swing/Icon.html), [Class.getResourceAsStream(String)](http://docs.google.com/java/lang/Class.html#getResourceAsStream(java.lang.String))

### getLayoutStyle

public [LayoutStyle](http://docs.google.com/javax/swing/LayoutStyle.html) **getLayoutStyle**()

Returns the LayoutStyle for this look and feel. This never returns null.

You generally don't use the LayoutStyle from the look and feel, instead use the LayoutStyle method getInstance.

**Returns:**the LayoutStyle for this look and feel**Since:** 1.6 **See Also:**[LayoutStyle.getInstance()](http://docs.google.com/javax/swing/LayoutStyle.html#getInstance())

### provideErrorFeedback

public void **provideErrorFeedback**([Component](http://docs.google.com/java/awt/Component.html) component)

Invoked when the user attempts an invalid operation, such as pasting into an uneditable JTextField that has focus. The default implementation beeps. Subclasses that wish different behavior should override this and provide the additional feedback.

**Parameters:**component - the Component the error occurred in, may be null indicating the error condition is not directly associated with a Component**Since:** 1.4

### getDesktopPropertyValue

public static [Object](http://docs.google.com/java/lang/Object.html) **getDesktopPropertyValue**([String](http://docs.google.com/java/lang/String.html) systemPropertyName,  
 [Object](http://docs.google.com/java/lang/Object.html) fallbackValue)

Returns the value of the specified system desktop property by invoking Toolkit.getDefaultToolkit().getDesktopProperty(). If the value of the specified property is null, fallbackValue is returned.

**Parameters:**systemPropertyName - the name of the system desktop property being queriedfallbackValue - the object to be returned as the value if the system value is null **Returns:**the current value of the desktop property**Since:** 1.4 **See Also:**[Toolkit.getDesktopProperty(java.lang.String)](http://docs.google.com/java/awt/Toolkit.html#getDesktopProperty(java.lang.String))

### getDisabledIcon

public [Icon](http://docs.google.com/javax/swing/Icon.html) **getDisabledIcon**([JComponent](http://docs.google.com/javax/swing/JComponent.html) component,  
 [Icon](http://docs.google.com/javax/swing/Icon.html) icon)

Returns an Icon with a disabled appearance. This method is used to generate a disabled Icon when one has not been specified. For example, if you create a JButton and only specify an Icon via setIcon this method will be called to generate the disabled Icon. If null is passed as icon this method returns null.

Some look and feels might not render the disabled Icon, in which case they will ignore this.

**Parameters:**component - JComponent that will display the Icon, may be nullicon - Icon to generate the disabled icon from **Returns:**disabled Icon, or null if a suitable Icon can not be generated**Since:** 1.5

### getDisabledSelectedIcon

public [Icon](http://docs.google.com/javax/swing/Icon.html) **getDisabledSelectedIcon**([JComponent](http://docs.google.com/javax/swing/JComponent.html) component,  
 [Icon](http://docs.google.com/javax/swing/Icon.html) icon)

Returns an Icon for use by disabled components that are also selected. This method is used to generate an Icon for components that are in both the disabled and selected states but do not have a specific Icon for this state. For example, if you create a JButton and only specify an Icon via setIcon this method will be called to generate the disabled and selected Icon. If null is passed as icon this methods returns null.

Some look and feels might not render the disabled and selected Icon, in which case they will ignore this.

**Parameters:**component - JComponent that will display the Icon, may be nullicon - Icon to generate disabled and selected icon from **Returns:**disabled and selected icon, or null if a suitable Icon can not be generated.**Since:** 1.5

### getName

public abstract [String](http://docs.google.com/java/lang/String.html) **getName**()

Return a short string that identifies this look and feel, e.g. "CDE/Motif". This string should be appropriate for a menu item. Distinct look and feels should have different names, e.g. a subclass of MotifLookAndFeel that changes the way a few components are rendered should be called "CDE/Motif My Way"; something that would be useful to a user trying to select a L&F from a list of names.

**Returns:**short identifier for the look and feel

### getID

public abstract [String](http://docs.google.com/java/lang/String.html) **getID**()

Return a string that identifies this look and feel. This string will be used by applications/services that want to recognize well known look and feel implementations. Presently the well known names are "Motif", "Windows", "Mac", "Metal". Note that a LookAndFeel derived from a well known superclass that doesn't make any fundamental changes to the look or feel shouldn't override this method.

**Returns:**identifier for the look and feel

### getDescription

public abstract [String](http://docs.google.com/java/lang/String.html) **getDescription**()

Return a one line description of this look and feel implementation, e.g. "The CDE/Motif Look and Feel". This string is intended for the user, e.g. in the title of a window or in a ToolTip message.

**Returns:**short description for the look and feel

### getSupportsWindowDecorations

public boolean **getSupportsWindowDecorations**()

Returns true if the LookAndFeel returned RootPaneUI instances support providing Window decorations in a JRootPane.

The default implementation returns false, subclasses that support Window decorations should override this and return true.

**Returns:**true if the RootPaneUI instances created by this look and feel support client side decorations**Since:** 1.4 **See Also:**[JDialog.setDefaultLookAndFeelDecorated(boolean)](http://docs.google.com/javax/swing/JDialog.html#setDefaultLookAndFeelDecorated(boolean)), [JFrame.setDefaultLookAndFeelDecorated(boolean)](http://docs.google.com/javax/swing/JFrame.html#setDefaultLookAndFeelDecorated(boolean)), [JRootPane.setWindowDecorationStyle(int)](http://docs.google.com/javax/swing/JRootPane.html#setWindowDecorationStyle(int))

### isNativeLookAndFeel

public abstract boolean **isNativeLookAndFeel**()

If the underlying platform has a "native" look and feel, and this is an implementation of it, return true. For example, when the underlying platform is Solaris running CDE a CDE/Motif look and feel implementation would return true.

**Returns:**true if this look and feel represents the underlying platform look and feel

### isSupportedLookAndFeel

public abstract boolean **isSupportedLookAndFeel**()

Return true if the underlying platform supports and or permits this look and feel. This method returns false if the look and feel depends on special resources or legal agreements that aren't defined for the current platform.

**Returns:**true if this is a supported look and feel**See Also:**[UIManager.setLookAndFeel(javax.swing.LookAndFeel)](http://docs.google.com/javax/swing/UIManager.html#setLookAndFeel(javax.swing.LookAndFeel))

### initialize

public void **initialize**()

Initializes the look and feel. While this method is public, it should only be invoked by the UIManager when a look and feel is installed as the current look and feel. This method is invoked before the UIManager invokes getDefaults. This method is intended to perform any initialization for the look and feel. Subclasses should do any one-time setup they need here, rather than in a static initializer, because look and feel class objects may be loaded just to discover that isSupportedLookAndFeel() returns false.

**See Also:**[uninitialize()](http://docs.google.com/javax/swing/LookAndFeel.html#uninitialize()), [UIManager.setLookAndFeel(javax.swing.LookAndFeel)](http://docs.google.com/javax/swing/UIManager.html#setLookAndFeel(javax.swing.LookAndFeel))

### uninitialize

public void **uninitialize**()

Uninitializes the look and feel. While this method is public, it should only be invoked by the UIManager when the look and feel is uninstalled. For example, UIManager.setLookAndFeel invokes this when the look and feel is changed.

Subclasses may choose to free up some resources here.

**See Also:**[initialize()](http://docs.google.com/javax/swing/LookAndFeel.html#initialize()), [UIManager.setLookAndFeel(javax.swing.LookAndFeel)](http://docs.google.com/javax/swing/UIManager.html#setLookAndFeel(javax.swing.LookAndFeel))

### getDefaults

public [UIDefaults](http://docs.google.com/javax/swing/UIDefaults.html) **getDefaults**()

Returns the look and feel defaults. While this method is public, it should only be invoked by the UIManager when the look and feel is set as the current look and feel and after initialize has been invoked.

**Returns:**the look and feel defaults**See Also:**[initialize()](http://docs.google.com/javax/swing/LookAndFeel.html#initialize()), [uninitialize()](http://docs.google.com/javax/swing/LookAndFeel.html#uninitialize()), [UIManager.setLookAndFeel(javax.swing.LookAndFeel)](http://docs.google.com/javax/swing/UIManager.html#setLookAndFeel(javax.swing.LookAndFeel))

### toString

public [String](http://docs.google.com/java/lang/String.html) **toString**()

Returns a string that displays and identifies this object's properties.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a String representation of this object

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/LookAndFeel.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/ListSelectionModel.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/MenuElement.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/LookAndFeel.html)    [**NO FRAMES**](http://docs.google.com/LookAndFeel.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: FIELD | [CONSTR](#4d34og8) | [METHOD](#17dp8vu) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).