| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/View.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/text/Utilities.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/text/ViewFactory.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/text/View.html)    [**NO FRAMES**](http://docs.google.com/View.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: [FIELD](#4d34og8) | [CONSTR](#1ksv4uv) | [METHOD](#2jxsxqh) |

## **javax.swing.text**

Class View

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

**All Implemented Interfaces:** [SwingConstants](http://docs.google.com/javax/swing/SwingConstants.html) **Direct Known Subclasses:** [AsyncBoxView](http://docs.google.com/javax/swing/text/AsyncBoxView.html), [ComponentView](http://docs.google.com/javax/swing/text/ComponentView.html), [CompositeView](http://docs.google.com/javax/swing/text/CompositeView.html), [GlyphView](http://docs.google.com/javax/swing/text/GlyphView.html), [IconView](http://docs.google.com/javax/swing/text/IconView.html), [ImageView](http://docs.google.com/javax/swing/text/html/ImageView.html), [PlainView](http://docs.google.com/javax/swing/text/PlainView.html)

public abstract class **View**extends [Object](http://docs.google.com/java/lang/Object.html)implements [SwingConstants](http://docs.google.com/javax/swing/SwingConstants.html)

A very important part of the text package is the View class. As the name suggests it represents a view of the text model, or a piece of the text model. It is this class that is responsible for the look of the text component. The view is not intended to be some completely new thing that one must learn, but rather is much like a lightweight component.

By default, a view is very light. It contains a reference to the parent view from which it can fetch many things without holding state, and it contains a reference to a portion of the model (Element). A view does not have to exactly represent an element in the model, that is simply a typical and therefore convenient mapping. A view can alternatively maintain a couple of Position objects to maintain its location in the model (i.e. represent a fragment of an element). This is typically the result of formatting where views have been broken down into pieces. The convenience of a substantial relationship to the element makes it easier to build factories to produce the views, and makes it easier to keep track of the view pieces as the model is changed and the view must be changed to reflect the model. Simple views therefore represent an Element directly and complex views do not.

A view has the following responsibilities:

**Participate in layout.**

The view has a setSize method which is like doLayout and setSize in Component combined. The view has a preferenceChanged method which is like invalidate in Component except that one can invalidate just one axis and the child requesting the change is identified.

A View expresses the size that it would like to be in terms of three values, a minimum, a preferred, and a maximum span. Layout in a view is can be done independently upon each axis. For a properly functioning View implementation, the minimum span will be <= the preferred span which in turn will be <= the maximum span.



The minimum set of methods for layout are:

* [getMinimumSpan](#4i7ojhp)
* [getPreferredSpan](#1y810tw)
* [getMaximumSpan](#2xcytpi)
* [getAlignment](#3whwml4)
* [preferenceChanged](#1ci93xb)
* [setSize](#3l18frh)

The setSize method should be prepared to be called a number of times (i.e. It may be called even if the size didn't change). The setSize method is generally called to make sure the View layout is complete prior to trying to perform an operation on it that requires an up-to-date layout. A view's size should *always* be set to a value within the minimum and maximum span specified by that view. Additionally, the view must always call the preferenceChanged method on the parent if it has changed the values for the layout it would like, and expects the parent to honor. The parent View is not required to recognize a change until the preferenceChanged has been sent. This allows parent View implementations to cache the child requirements if desired. The calling sequence looks something like the following:



The exact calling sequence is up to the layout functionality of the parent view (if the view has any children). The view may collect the preferences of the children prior to determining what it will give each child, or it might iteratively update the children one at a time.

**Render a portion of the model.**

This is done in the paint method, which is pretty much like a component paint method. Views are expected to potentially populate a fairly large tree. A View has the following semantics for rendering:

* The view gets its allocation from the parent at paint time, so it must be prepared to redo layout if the allocated area is different from what it is prepared to deal with.
* The coordinate system is the same as the hosting Component (i.e. the Component returned by the [getContainer](#25b2l0r) method). This means a child view lives in the same coordinate system as the parent view unless the parent has explicitly changed the coordinate system. To schedule itself to be repainted a view can call repaint on the hosting Component.
* The default is to *not clip* the children. It is more efficient to allow a view to clip only if it really feels it needs clipping.
* The Graphics object given is not initialized in any way. A view should set any settings needed.
* A View is inherently transparent. While a view may render into its entire allocation, typically a view does not. Rendering is performed by tranversing down the tree of View implementations. Each View is responsible for rendering its children. This behavior is depended upon for thread safety. While view implementations do not necessarily have to be implemented with thread safety in mind, other view implementations that do make use of concurrency can depend upon a tree traversal to guarantee thread safety.
* The order of views relative to the model is up to the implementation. Although child views will typically be arranged in the same order that they occur in the model, they may be visually arranged in an entirely different order. View implementations may have Z-Order associated with them if the children are overlapping.

The methods for rendering are:

* [paint](#2bn6wsx)

**Translate between the model and view coordinate systems.**

Because the view objects are produced from a factory and therefore cannot necessarily be counted upon to be in a particular pattern, one must be able to perform translation to properly locate spatial representation of the model. The methods for doing this are:

* [modelToView](#2grqrue)
* [viewToModel](#vx1227)
* [getDocument](#2u6wntf)
* [getElement](#28h4qwu)
* [getStartOffset](#19c6y18)
* [getEndOffset](#3tbugp1)

The layout must be valid prior to attempting to make the translation. The translation is not valid, and must not be attempted while changes are being broadcasted from the model via a DocumentEvent.

**Respond to changes from the model.**

If the overall view is represented by many pieces (which is the best situation if one want to be able to change the view and write the least amount of new code), it would be impractical to have a huge number of DocumentListeners. If each view listened to the model, only a few would actually be interested in the changes broadcasted at any given time. Since the model has no knowledge of views, it has no way to filter the broadcast of change information. The view hierarchy itself is instead responsible for propagating the change information. At any level in the view hierarchy, that view knows enough about its children to best distribute the change information further. Changes are therefore broadcasted starting from the root of the view hierarchy. The methods for doing this are:

* [insertUpdate](#kgcv8k)
* [removeUpdate](#34g0dwd)
* [changedUpdate](#1jlao46)

| **Field Summary** | |
| --- | --- |
| static int | [**BadBreakWeight**](http://docs.google.com/javax/swing/text/View.html#BadBreakWeight)            The weight to indicate a view is a bad break opportunity for the purpose of formatting. |
| static int | [**ExcellentBreakWeight**](http://docs.google.com/javax/swing/text/View.html#ExcellentBreakWeight)            The weight to indicate a view supports breaking, and this represents a very attractive place to break. |
| static int | [**ForcedBreakWeight**](http://docs.google.com/javax/swing/text/View.html#ForcedBreakWeight)            The weight to indicate a view supports breaking, and must be broken to be represented properly when placed in a view that formats its children by breaking them. |
| static int | [**GoodBreakWeight**](http://docs.google.com/javax/swing/text/View.html#GoodBreakWeight)            The weight to indicate a view supports breaking, but better opportunities probably exist. |
| static int | [**X\_AXIS**](http://docs.google.com/javax/swing/text/View.html#X_AXIS)            Axis for format/break operations. |
| static int | [**Y\_AXIS**](http://docs.google.com/javax/swing/text/View.html#Y_AXIS)            Axis for format/break operations. |

| **Fields inherited from interface javax.swing.**[**SwingConstants**](http://docs.google.com/javax/swing/SwingConstants.html) |
| --- |
| [BOTTOM](http://docs.google.com/javax/swing/SwingConstants.html#BOTTOM), [CENTER](http://docs.google.com/javax/swing/SwingConstants.html#CENTER), [EAST](http://docs.google.com/javax/swing/SwingConstants.html#EAST), [HORIZONTAL](http://docs.google.com/javax/swing/SwingConstants.html#HORIZONTAL), [LEADING](http://docs.google.com/javax/swing/SwingConstants.html#LEADING), [LEFT](http://docs.google.com/javax/swing/SwingConstants.html#LEFT), [NEXT](http://docs.google.com/javax/swing/SwingConstants.html#NEXT), [NORTH](http://docs.google.com/javax/swing/SwingConstants.html#NORTH), [NORTH\_EAST](http://docs.google.com/javax/swing/SwingConstants.html#NORTH_EAST), [NORTH\_WEST](http://docs.google.com/javax/swing/SwingConstants.html#NORTH_WEST), [PREVIOUS](http://docs.google.com/javax/swing/SwingConstants.html#PREVIOUS), [RIGHT](http://docs.google.com/javax/swing/SwingConstants.html#RIGHT), [SOUTH](http://docs.google.com/javax/swing/SwingConstants.html#SOUTH), [SOUTH\_EAST](http://docs.google.com/javax/swing/SwingConstants.html#SOUTH_EAST), [SOUTH\_WEST](http://docs.google.com/javax/swing/SwingConstants.html#SOUTH_WEST), [TOP](http://docs.google.com/javax/swing/SwingConstants.html#TOP), [TRAILING](http://docs.google.com/javax/swing/SwingConstants.html#TRAILING), [VERTICAL](http://docs.google.com/javax/swing/SwingConstants.html#VERTICAL), [WEST](http://docs.google.com/javax/swing/SwingConstants.html#WEST) |

| **Constructor Summary** | |
| --- | --- |
| [**View**](http://docs.google.com/javax/swing/text/View.html#View(javax.swing.text.Element))([Element](http://docs.google.com/javax/swing/text/Element.html) elem)            Creates a new View object. |

| **Method Summary** | |
| --- | --- |
| void | [**append**](http://docs.google.com/javax/swing/text/View.html#append(javax.swing.text.View))([View](http://docs.google.com/javax/swing/text/View.html) v)            Appends a single child view. |
| [View](http://docs.google.com/javax/swing/text/View.html) | [**breakView**](http://docs.google.com/javax/swing/text/View.html#breakView(int,%20int,%20float,%20float))(int axis, int offset, float pos, float len)            Tries to break this view on the given axis. |
| void | [**changedUpdate**](http://docs.google.com/javax/swing/text/View.html#changedUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))([DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e, [Shape](http://docs.google.com/java/awt/Shape.html) a, [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)            Gives notification from the document that attributes were changed in a location that this view is responsible for. |
| [View](http://docs.google.com/javax/swing/text/View.html) | [**createFragment**](http://docs.google.com/javax/swing/text/View.html#createFragment(int,%20int))(int p0, int p1)            Creates a view that represents a portion of the element. |
| protected  void | [**forwardUpdate**](http://docs.google.com/javax/swing/text/View.html#forwardUpdate(javax.swing.event.DocumentEvent.ElementChange,%20javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))([DocumentEvent.ElementChange](http://docs.google.com/javax/swing/event/DocumentEvent.ElementChange.html) ec, [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e, [Shape](http://docs.google.com/java/awt/Shape.html) a, [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)            Forwards the given DocumentEvent to the child views that need to be notified of the change to the model. |
| protected  void | [**forwardUpdateToView**](http://docs.google.com/javax/swing/text/View.html#forwardUpdateToView(javax.swing.text.View,%20javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))([View](http://docs.google.com/javax/swing/text/View.html) v, [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e, [Shape](http://docs.google.com/java/awt/Shape.html) a, [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)            Forwards the DocumentEvent to the give child view. |
| float | [**getAlignment**](http://docs.google.com/javax/swing/text/View.html#getAlignment(int))(int axis)            Determines the desired alignment for this view along an axis. |
| [AttributeSet](http://docs.google.com/javax/swing/text/AttributeSet.html) | [**getAttributes**](http://docs.google.com/javax/swing/text/View.html#getAttributes())()            Fetches the attributes to use when rendering. |
| int | [**getBreakWeight**](http://docs.google.com/javax/swing/text/View.html#getBreakWeight(int,%20float,%20float))(int axis, float pos, float len)            Determines how attractive a break opportunity in this view is. |
| [Shape](http://docs.google.com/java/awt/Shape.html) | [**getChildAllocation**](http://docs.google.com/javax/swing/text/View.html#getChildAllocation(int,%20java.awt.Shape))(int index, [Shape](http://docs.google.com/java/awt/Shape.html) a)            Fetches the allocation for the given child view. |
| [Container](http://docs.google.com/java/awt/Container.html) | [**getContainer**](http://docs.google.com/javax/swing/text/View.html#getContainer())()            Fetches the container hosting the view. |
| [Document](http://docs.google.com/javax/swing/text/Document.html) | [**getDocument**](http://docs.google.com/javax/swing/text/View.html#getDocument())()            Fetches the model associated with the view. |
| [Element](http://docs.google.com/javax/swing/text/Element.html) | [**getElement**](http://docs.google.com/javax/swing/text/View.html#getElement())()            Fetches the structural portion of the subject that this view is mapped to. |
| int | [**getEndOffset**](http://docs.google.com/javax/swing/text/View.html#getEndOffset())()            Fetches the portion of the model for which this view is responsible. |
| [Graphics](http://docs.google.com/java/awt/Graphics.html) | [**getGraphics**](http://docs.google.com/javax/swing/text/View.html#getGraphics())()            Fetch a Graphics for rendering. |
| float | [**getMaximumSpan**](http://docs.google.com/javax/swing/text/View.html#getMaximumSpan(int))(int axis)            Determines the maximum span for this view along an axis. |
| float | [**getMinimumSpan**](http://docs.google.com/javax/swing/text/View.html#getMinimumSpan(int))(int axis)            Determines the minimum span for this view along an axis. |
| int | [**getNextVisualPositionFrom**](http://docs.google.com/javax/swing/text/View.html#getNextVisualPositionFrom(int,%20javax.swing.text.Position.Bias,%20java.awt.Shape,%20int,%20javax.swing.text.Position.Bias%5B%5D))(int pos, [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b, [Shape](http://docs.google.com/java/awt/Shape.html) a, int direction, [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html)[] biasRet)            Provides a way to determine the next visually represented model location at which one might place a caret. |
| [View](http://docs.google.com/javax/swing/text/View.html) | [**getParent**](http://docs.google.com/javax/swing/text/View.html#getParent())()            Returns the parent of the view. |
| abstract  float | [**getPreferredSpan**](http://docs.google.com/javax/swing/text/View.html#getPreferredSpan(int))(int axis)            Determines the preferred span for this view along an axis. |
| int | [**getResizeWeight**](http://docs.google.com/javax/swing/text/View.html#getResizeWeight(int))(int axis)            Determines the resizability of the view along the given axis. |
| int | [**getStartOffset**](http://docs.google.com/javax/swing/text/View.html#getStartOffset())()            Fetches the portion of the model for which this view is responsible. |
| [String](http://docs.google.com/java/lang/String.html) | [**getToolTipText**](http://docs.google.com/javax/swing/text/View.html#getToolTipText(float,%20float,%20java.awt.Shape))(float x, float y, [Shape](http://docs.google.com/java/awt/Shape.html) allocation)            Returns the tooltip text at the specified location. |
| [View](http://docs.google.com/javax/swing/text/View.html) | [**getView**](http://docs.google.com/javax/swing/text/View.html#getView(int))(int n)            Gets the *n*th child view. |
| int | [**getViewCount**](http://docs.google.com/javax/swing/text/View.html#getViewCount())()            Returns the number of views in this view. |
| [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) | [**getViewFactory**](http://docs.google.com/javax/swing/text/View.html#getViewFactory())()            Fetches the ViewFactory implementation that is feeding the view hierarchy. |
| int | [**getViewIndex**](http://docs.google.com/javax/swing/text/View.html#getViewIndex(float,%20float,%20java.awt.Shape))(float x, float y, [Shape](http://docs.google.com/java/awt/Shape.html) allocation)            Returns the child view index representing the given position in the view. |
| int | [**getViewIndex**](http://docs.google.com/javax/swing/text/View.html#getViewIndex(int,%20javax.swing.text.Position.Bias))(int pos, [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b)            Returns the child view index representing the given position in the model. |
| void | [**insert**](http://docs.google.com/javax/swing/text/View.html#insert(int,%20javax.swing.text.View))(int offs, [View](http://docs.google.com/javax/swing/text/View.html) v)            Inserts a single child view. |
| void | [**insertUpdate**](http://docs.google.com/javax/swing/text/View.html#insertUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))([DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e, [Shape](http://docs.google.com/java/awt/Shape.html) a, [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)            Gives notification that something was inserted into the document in a location that this view is responsible for. |
| boolean | [**isVisible**](http://docs.google.com/javax/swing/text/View.html#isVisible())()            Returns a boolean that indicates whether the view is visible or not. |
| [Shape](http://docs.google.com/java/awt/Shape.html) | [**modelToView**](http://docs.google.com/javax/swing/text/View.html#modelToView(int,%20javax.swing.text.Position.Bias,%20int,%20javax.swing.text.Position.Bias,%20java.awt.Shape))(int p0, [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b0, int p1, [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b1, [Shape](http://docs.google.com/java/awt/Shape.html) a)            Provides a mapping, for a given region, from the document model coordinate space to the view coordinate space. |
| [Shape](http://docs.google.com/java/awt/Shape.html) | [**modelToView**](http://docs.google.com/javax/swing/text/View.html#modelToView(int,%20java.awt.Shape))(int pos, [Shape](http://docs.google.com/java/awt/Shape.html) a)  **Deprecated.** |
| abstract  [Shape](http://docs.google.com/java/awt/Shape.html) | [**modelToView**](http://docs.google.com/javax/swing/text/View.html#modelToView(int,%20java.awt.Shape,%20javax.swing.text.Position.Bias))(int pos, [Shape](http://docs.google.com/java/awt/Shape.html) a, [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b)            Provides a mapping, for a given character, from the document model coordinate space to the view coordinate space. |
| abstract  void | [**paint**](http://docs.google.com/javax/swing/text/View.html#paint(java.awt.Graphics,%20java.awt.Shape))([Graphics](http://docs.google.com/java/awt/Graphics.html) g, [Shape](http://docs.google.com/java/awt/Shape.html) allocation)            Renders using the given rendering surface and area on that surface. |
| void | [**preferenceChanged**](http://docs.google.com/javax/swing/text/View.html#preferenceChanged(javax.swing.text.View,%20boolean,%20boolean))([View](http://docs.google.com/javax/swing/text/View.html) child, boolean width, boolean height)            Child views can call this on the parent to indicate that the preference has changed and should be reconsidered for layout. |
| void | [**remove**](http://docs.google.com/javax/swing/text/View.html#remove(int))(int i)            Removes one of the children at the given position. |
| void | [**removeAll**](http://docs.google.com/javax/swing/text/View.html#removeAll())()            Removes all of the children. |
| void | [**removeUpdate**](http://docs.google.com/javax/swing/text/View.html#removeUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))([DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e, [Shape](http://docs.google.com/java/awt/Shape.html) a, [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)            Gives notification that something was removed from the document in a location that this view is responsible for. |
| void | [**replace**](http://docs.google.com/javax/swing/text/View.html#replace(int,%20int,%20javax.swing.text.View%5B%5D))(int offset, int length, [View](http://docs.google.com/javax/swing/text/View.html)[] views)            Replaces child views. |
| void | [**setParent**](http://docs.google.com/javax/swing/text/View.html#setParent(javax.swing.text.View))([View](http://docs.google.com/javax/swing/text/View.html) parent)            Establishes the parent view for this view. |
| void | [**setSize**](http://docs.google.com/javax/swing/text/View.html#setSize(float,%20float))(float width, float height)            Sets the size of the view. |
| protected  boolean | [**updateChildren**](http://docs.google.com/javax/swing/text/View.html#updateChildren(javax.swing.event.DocumentEvent.ElementChange,%20javax.swing.event.DocumentEvent,%20javax.swing.text.ViewFactory))([DocumentEvent.ElementChange](http://docs.google.com/javax/swing/event/DocumentEvent.ElementChange.html) ec, [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e, [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)            Updates the child views in response to receiving notification that the model changed, and there is change record for the element this view is responsible for. |
| protected  void | [**updateLayout**](http://docs.google.com/javax/swing/text/View.html#updateLayout(javax.swing.event.DocumentEvent.ElementChange,%20javax.swing.event.DocumentEvent,%20java.awt.Shape))([DocumentEvent.ElementChange](http://docs.google.com/javax/swing/event/DocumentEvent.ElementChange.html) ec, [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e, [Shape](http://docs.google.com/java/awt/Shape.html) a)            Updates the layout in response to receiving notification of change from the model. |
| int | [**viewToModel**](http://docs.google.com/javax/swing/text/View.html#viewToModel(float,%20float,%20java.awt.Shape))(float x, float y, [Shape](http://docs.google.com/java/awt/Shape.html) a)  **Deprecated.** |
| abstract  int | [**viewToModel**](http://docs.google.com/javax/swing/text/View.html#viewToModel(float,%20float,%20java.awt.Shape,%20javax.swing.text.Position.Bias%5B%5D))(float x, float y, [Shape](http://docs.google.com/java/awt/Shape.html) a, [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html)[] biasReturn)            Provides a mapping from the view coordinate space to the logical coordinate space of the model. |

| **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()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [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)) |

| **Field Detail** |
| --- |

### BadBreakWeight

public static final int **BadBreakWeight**

The weight to indicate a view is a bad break opportunity for the purpose of formatting. This value indicates that no attempt should be made to break the view into fragments as the view has not been written to support fragmenting.

**See Also:**[getBreakWeight(int, float, float)](http://docs.google.com/javax/swing/text/View.html#getBreakWeight(int,%20float,%20float)), [GoodBreakWeight](http://docs.google.com/javax/swing/text/View.html#GoodBreakWeight), [ExcellentBreakWeight](http://docs.google.com/javax/swing/text/View.html#ExcellentBreakWeight), [ForcedBreakWeight](http://docs.google.com/javax/swing/text/View.html#ForcedBreakWeight), [Constant Field Values](http://docs.google.com/constant-values.html#javax.swing.text.View.BadBreakWeight)

### GoodBreakWeight

public static final int **GoodBreakWeight**

The weight to indicate a view supports breaking, but better opportunities probably exist.

**See Also:**[getBreakWeight(int, float, float)](http://docs.google.com/javax/swing/text/View.html#getBreakWeight(int,%20float,%20float)), [BadBreakWeight](http://docs.google.com/javax/swing/text/View.html#BadBreakWeight), [ExcellentBreakWeight](http://docs.google.com/javax/swing/text/View.html#ExcellentBreakWeight), [ForcedBreakWeight](http://docs.google.com/javax/swing/text/View.html#ForcedBreakWeight), [Constant Field Values](http://docs.google.com/constant-values.html#javax.swing.text.View.GoodBreakWeight)

### ExcellentBreakWeight

public static final int **ExcellentBreakWeight**

The weight to indicate a view supports breaking, and this represents a very attractive place to break.

**See Also:**[getBreakWeight(int, float, float)](http://docs.google.com/javax/swing/text/View.html#getBreakWeight(int,%20float,%20float)), [BadBreakWeight](http://docs.google.com/javax/swing/text/View.html#BadBreakWeight), [GoodBreakWeight](http://docs.google.com/javax/swing/text/View.html#GoodBreakWeight), [ForcedBreakWeight](http://docs.google.com/javax/swing/text/View.html#ForcedBreakWeight), [Constant Field Values](http://docs.google.com/constant-values.html#javax.swing.text.View.ExcellentBreakWeight)

### ForcedBreakWeight

public static final int **ForcedBreakWeight**

The weight to indicate a view supports breaking, and must be broken to be represented properly when placed in a view that formats its children by breaking them.

**See Also:**[getBreakWeight(int, float, float)](http://docs.google.com/javax/swing/text/View.html#getBreakWeight(int,%20float,%20float)), [BadBreakWeight](http://docs.google.com/javax/swing/text/View.html#BadBreakWeight), [GoodBreakWeight](http://docs.google.com/javax/swing/text/View.html#GoodBreakWeight), [ExcellentBreakWeight](http://docs.google.com/javax/swing/text/View.html#ExcellentBreakWeight), [Constant Field Values](http://docs.google.com/constant-values.html#javax.swing.text.View.ForcedBreakWeight)

### X\_AXIS

public static final int **X\_AXIS**

Axis for format/break operations.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.swing.text.View.X_AXIS)

### Y\_AXIS

public static final int **Y\_AXIS**

Axis for format/break operations.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.swing.text.View.Y_AXIS)

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

### View

public **View**([Element](http://docs.google.com/javax/swing/text/Element.html) elem)

Creates a new View object.

**Parameters:**elem - the Element to represent

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

### getParent

public [View](http://docs.google.com/javax/swing/text/View.html) **getParent**()

Returns the parent of the view.

**Returns:**the parent, or null if none exists

### isVisible

public boolean **isVisible**()

Returns a boolean that indicates whether the view is visible or not. By default all views are visible.

**Returns:**always returns true

### getPreferredSpan

public abstract float **getPreferredSpan**(int axis)

Determines the preferred span for this view along an axis.

**Parameters:**axis - may be either View.X\_AXIS or View.Y\_AXIS **Returns:**the span the view would like to be rendered into. Typically the view is told to render into the span that is returned, although there is no guarantee. The parent may choose to resize or break the view**See Also:**[getPreferredSpan(int)](http://docs.google.com/javax/swing/text/View.html#getPreferredSpan(int))

### getMinimumSpan

public float **getMinimumSpan**(int axis)

Determines the minimum span for this view along an axis.

**Parameters:**axis - may be either View.X\_AXIS or View.Y\_AXIS **Returns:**the minimum span the view can be rendered into**See Also:**[getPreferredSpan(int)](http://docs.google.com/javax/swing/text/View.html#getPreferredSpan(int))

### getMaximumSpan

public float **getMaximumSpan**(int axis)

Determines the maximum span for this view along an axis.

**Parameters:**axis - may be either View.X\_AXIS or View.Y\_AXIS **Returns:**the maximum span the view can be rendered into**See Also:**[getPreferredSpan(int)](http://docs.google.com/javax/swing/text/View.html#getPreferredSpan(int))

### preferenceChanged

public void **preferenceChanged**([View](http://docs.google.com/javax/swing/text/View.html) child,  
 boolean width,  
 boolean height)

Child views can call this on the parent to indicate that the preference has changed and should be reconsidered for layout. By default this just propagates upward to the next parent. The root view will call revalidate on the associated text component.

**Parameters:**child - the child viewwidth - true if the width preference has changedheight - true if the height preference has changed**See Also:**[JComponent.revalidate()](http://docs.google.com/javax/swing/JComponent.html#revalidate())

### getAlignment

public float **getAlignment**(int axis)

Determines the desired alignment for this view along an axis. The desired alignment is returned. This should be a value >= 0.0 and <= 1.0, where 0 indicates alignment at the origin and 1.0 indicates alignment to the full span away from the origin. An alignment of 0.5 would be the center of the view.

**Parameters:**axis - may be either View.X\_AXIS or View.Y\_AXIS **Returns:**the value 0.5

### paint

public abstract void **paint**([Graphics](http://docs.google.com/java/awt/Graphics.html) g,  
 [Shape](http://docs.google.com/java/awt/Shape.html) allocation)

Renders using the given rendering surface and area on that surface. The view may need to do layout and create child views to enable itself to render into the given allocation.

**Parameters:**g - the rendering surface to useallocation - the allocated region to render into

### setParent

public void **setParent**([View](http://docs.google.com/javax/swing/text/View.html) parent)

Establishes the parent view for this view. This is guaranteed to be called before any other methods if the parent view is functioning properly. This is also the last method called, since it is called to indicate the view has been removed from the hierarchy as well. When this method is called to set the parent to null, this method does the same for each of its children, propogating the notification that they have been disconnected from the view tree. If this is reimplemented, super.setParent() should be called.

**Parameters:**parent - the new parent, or null if the view is being removed from a parent

### getViewCount

public int **getViewCount**()

Returns the number of views in this view. Since the default is to not be a composite view this returns 0.

**Returns:**the number of views >= 0**See Also:**[getViewCount()](http://docs.google.com/javax/swing/text/View.html#getViewCount())

### getView

public [View](http://docs.google.com/javax/swing/text/View.html) **getView**(int n)

Gets the *n*th child view. Since there are no children by default, this returns null.

**Parameters:**n - the number of the view to get, >= 0 && < getViewCount() **Returns:**the view

### removeAll

public void **removeAll**()

Removes all of the children. This is a convenience call to replace.

**Since:** 1.3

### remove

public void **remove**(int i)

Removes one of the children at the given position. This is a convenience call to replace.

**Since:** 1.3

### insert

public void **insert**(int offs,  
 [View](http://docs.google.com/javax/swing/text/View.html) v)

Inserts a single child view. This is a convenience call to replace.

**Parameters:**offs - the offset of the view to insert before >= 0v - the view**Since:** 1.3 **See Also:**[replace(int, int, javax.swing.text.View[])](http://docs.google.com/javax/swing/text/View.html#replace(int,%20int,%20javax.swing.text.View%5B%5D))

### append

public void **append**([View](http://docs.google.com/javax/swing/text/View.html) v)

Appends a single child view. This is a convenience call to replace.

**Parameters:**v - the view**Since:** 1.3 **See Also:**[replace(int, int, javax.swing.text.View[])](http://docs.google.com/javax/swing/text/View.html#replace(int,%20int,%20javax.swing.text.View%5B%5D))

### replace

public void **replace**(int offset,  
 int length,  
 [View](http://docs.google.com/javax/swing/text/View.html)[] views)

Replaces child views. If there are no views to remove this acts as an insert. If there are no views to add this acts as a remove. Views being removed will have the parent set to null, and the internal reference to them removed so that they can be garbage collected. This is implemented to do nothing, because by default a view has no children.

**Parameters:**offset - the starting index into the child views to insert the new views. This should be a value >= 0 and <= getViewCountlength - the number of existing child views to remove This should be a value >= 0 and <= (getViewCount() - offset).views - the child views to add. This value can be null to indicate no children are being added (useful to remove).**Since:** 1.3

### getViewIndex

public int **getViewIndex**(int pos,  
 [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b)

Returns the child view index representing the given position in the model. By default a view has no children so this is implemented to return -1 to indicate there is no valid child index for any position.

**Parameters:**pos - the position >= 0 **Returns:**index of the view representing the given position, or -1 if no view represents that position**Since:** 1.3

### getChildAllocation

public [Shape](http://docs.google.com/java/awt/Shape.html) **getChildAllocation**(int index,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a)

Fetches the allocation for the given child view. This enables finding out where various views are located, without assuming how the views store their location. This returns null since the default is to not have any child views.

**Parameters:**index - the index of the child, >= 0 && < getViewCount()a - the allocation to this view **Returns:**the allocation to the child

### getNextVisualPositionFrom

public int **getNextVisualPositionFrom**(int pos,  
 [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 int direction,  
 [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html)[] biasRet)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

Provides a way to determine the next visually represented model location at which one might place a caret. Some views may not be visible, they might not be in the same order found in the model, or they just might not allow access to some of the locations in the model.

**Parameters:**pos - the position to convert >= 0a - the allocated region in which to renderdirection - the direction from the current position that can be thought of as the arrow keys typically found on a keyboard. This will be one of the following values:

* SwingConstants.WEST
* SwingConstants.EAST
* SwingConstants.NORTH
* SwingConstants.SOUTH

**Returns:**the location within the model that best represents the next location visual position **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if direction doesn't have one of the legal values above

### modelToView

public abstract [Shape](http://docs.google.com/java/awt/Shape.html) **modelToView**(int pos,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

Provides a mapping, for a given character, from the document model coordinate space to the view coordinate space.

**Parameters:**pos - the position of the desired character (>=0)a - the area of the view, which encompasses the requested characterb - the bias toward the previous character or the next character represented by the offset, in case the position is a boundary of two views; b will have one of these values:

* Position.Bias.Forward
* Position.Bias.Backward

**Returns:**the bounding box, in view coordinate space, of the character at the specified position **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) - if the specified position does not represent a valid location in the associated document [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if b is not one of the legal Position.Bias values listed above**See Also:**[viewToModel(float, float, java.awt.Shape, javax.swing.text.Position.Bias[])](http://docs.google.com/javax/swing/text/View.html#viewToModel(float,%20float,%20java.awt.Shape,%20javax.swing.text.Position.Bias%5B%5D))

### modelToView

public [Shape](http://docs.google.com/java/awt/Shape.html) **modelToView**(int p0,  
 [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b0,  
 int p1,  
 [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html) b1,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

Provides a mapping, for a given region, from the document model coordinate space to the view coordinate space. The specified region is created as a union of the first and last character positions.

**Parameters:**p0 - the position of the first character (>=0)b0 - the bias of the first character position, toward the previous character or the next character represented by the offset, in case the position is a boundary of two views; b0 will have one of these values:

* Position.Bias.Forward
* Position.Bias.Backward

p1 - the position of the last character (>=0)b1 - the bias for the second character position, defined one of the legal values shown abovea - the area of the view, which encompasses the requested region **Returns:**the bounding box which is a union of the region specified by the first and last character positions **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) - if the given position does not represent a valid location in the associated document [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if b0 or b1 are not one of the legal Position.Bias values listed above**See Also:**[viewToModel(float, float, java.awt.Shape, javax.swing.text.Position.Bias[])](http://docs.google.com/javax/swing/text/View.html#viewToModel(float,%20float,%20java.awt.Shape,%20javax.swing.text.Position.Bias%5B%5D))

### viewToModel

public abstract int **viewToModel**(float x,  
 float y,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 [Position.Bias](http://docs.google.com/javax/swing/text/Position.Bias.html)[] biasReturn)

Provides a mapping from the view coordinate space to the logical coordinate space of the model. The biasReturn argument will be filled in to indicate that the point given is closer to the next character in the model or the previous character in the model.

**Parameters:**x - the X coordinate >= 0y - the Y coordinate >= 0a - the allocated region in which to render **Returns:**the location within the model that best represents the given point in the view >= 0. The biasReturn argument will be filled in to indicate that the point given is closer to the next character in the model or the previous character in the model.

### insertUpdate

public void **insertUpdate**([DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)

Gives notification that something was inserted into the document in a location that this view is responsible for. To reduce the burden to subclasses, this functionality is spread out into the following calls that subclasses can reimplement:

1. [updateChildren](#43ky6rz) is called if there were any changes to the element this view is responsible for. If this view has child views that are represent the child elements, then this method should do whatever is necessary to make sure the child views correctly represent the model.
2. [forwardUpdate](#2iq8gzs) is called to forward the DocumentEvent to the appropriate child views.
3. [updateLayout](#xvir7l) is called to give the view a chance to either repair its layout, to reschedule layout, or do nothing.

**Parameters:**e - the change information from the associated documenta - the current allocation of the viewf - the factory to use to rebuild if the view has children**See Also:**[insertUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#insertUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))

### removeUpdate

public void **removeUpdate**([DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)

Gives notification that something was removed from the document in a location that this view is responsible for. To reduce the burden to subclasses, this functionality is spread out into the following calls that subclasses can reimplement:

1. [updateChildren](#43ky6rz) is called if there were any changes to the element this view is responsible for. If this view has child views that are represent the child elements, then this method should do whatever is necessary to make sure the child views correctly represent the model.
2. [forwardUpdate](#2iq8gzs) is called to forward the DocumentEvent to the appropriate child views.
3. [updateLayout](#xvir7l) is called to give the view a chance to either repair its layout, to reschedule layout, or do nothing.

**Parameters:**e - the change information from the associated documenta - the current allocation of the viewf - the factory to use to rebuild if the view has children**See Also:**[removeUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#removeUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))

### changedUpdate

public void **changedUpdate**([DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)

Gives notification from the document that attributes were changed in a location that this view is responsible for. To reduce the burden to subclasses, this functionality is spread out into the following calls that subclasses can reimplement:

1. [updateChildren](#43ky6rz) is called if there were any changes to the element this view is responsible for. If this view has child views that are represent the child elements, then this method should do whatever is necessary to make sure the child views correctly represent the model.
2. [forwardUpdate](#2iq8gzs) is called to forward the DocumentEvent to the appropriate child views.
3. [updateLayout](#xvir7l) is called to give the view a chance to either repair its layout, to reschedule layout, or do nothing.

**Parameters:**e - the change information from the associated documenta - the current allocation of the viewf - the factory to use to rebuild if the view has children**See Also:**[changedUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#changedUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))

### getDocument

public [Document](http://docs.google.com/javax/swing/text/Document.html) **getDocument**()

Fetches the model associated with the view.

**Returns:**the view model, null if none**See Also:**[getDocument()](http://docs.google.com/javax/swing/text/View.html#getDocument())

### getStartOffset

public int **getStartOffset**()

Fetches the portion of the model for which this view is responsible.

**Returns:**the starting offset into the model >= 0**See Also:**[getStartOffset()](http://docs.google.com/javax/swing/text/View.html#getStartOffset())

### getEndOffset

public int **getEndOffset**()

Fetches the portion of the model for which this view is responsible.

**Returns:**the ending offset into the model >= 0**See Also:**[getEndOffset()](http://docs.google.com/javax/swing/text/View.html#getEndOffset())

### getElement

public [Element](http://docs.google.com/javax/swing/text/Element.html) **getElement**()

Fetches the structural portion of the subject that this view is mapped to. The view may not be responsible for the entire portion of the element.

**Returns:**the subject**See Also:**[getElement()](http://docs.google.com/javax/swing/text/View.html#getElement())

### getGraphics

public [Graphics](http://docs.google.com/java/awt/Graphics.html) **getGraphics**()

Fetch a Graphics for rendering. This can be used to determine font characteristics, and will be different for a print view than a component view.

**Returns:**a Graphics object for rendering**Since:** 1.3

### getAttributes

public [AttributeSet](http://docs.google.com/javax/swing/text/AttributeSet.html) **getAttributes**()

Fetches the attributes to use when rendering. By default this simply returns the attributes of the associated element. This method should be used rather than using the element directly to obtain access to the attributes to allow view-specific attributes to be mixed in or to allow the view to have view-specific conversion of attributes by subclasses. Each view should document what attributes it recognizes for the purpose of rendering or layout, and should always access them through the AttributeSet returned by this method.

### breakView

public [View](http://docs.google.com/javax/swing/text/View.html) **breakView**(int axis,  
 int offset,  
 float pos,  
 float len)

Tries to break this view on the given axis. This is called by views that try to do formatting of their children. For example, a view of a paragraph will typically try to place its children into row and views representing chunks of text can sometimes be broken down into smaller pieces.

This is implemented to return the view itself, which represents the default behavior on not being breakable. If the view does support breaking, the starting offset of the view returned should be the given offset, and the end offset should be less than or equal to the end offset of the view being broken.

**Parameters:**axis - may be either View.X\_AXIS or View.Y\_AXISoffset - the location in the document model that a broken fragment would occupy >= 0. This would be the starting offset of the fragment returnedpos - the position along the axis that the broken view would occupy >= 0. This may be useful for things like tab calculationslen - specifies the distance along the axis where a potential break is desired >= 0 **Returns:**the fragment of the view that represents the given span, if the view can be broken. If the view doesn't support breaking behavior, the view itself is returned.**See Also:**[ParagraphView](http://docs.google.com/javax/swing/text/ParagraphView.html)

### createFragment

public [View](http://docs.google.com/javax/swing/text/View.html) **createFragment**(int p0,  
 int p1)

Creates a view that represents a portion of the element. This is potentially useful during formatting operations for taking measurements of fragments of the view. If the view doesn't support fragmenting (the default), it should return itself.

**Parameters:**p0 - the starting offset >= 0. This should be a value greater or equal to the element starting offset and less than the element ending offset.p1 - the ending offset > p0. This should be a value less than or equal to the elements end offset and greater than the elements starting offset. **Returns:**the view fragment, or itself if the view doesn't support breaking into fragments**See Also:**[LabelView](http://docs.google.com/javax/swing/text/LabelView.html)

### getBreakWeight

public int **getBreakWeight**(int axis,  
 float pos,  
 float len)

Determines how attractive a break opportunity in this view is. This can be used for determining which view is the most attractive to call breakView on in the process of formatting. A view that represents text that has whitespace in it might be more attractive than a view that has no whitespace, for example. The higher the weight, the more attractive the break. A value equal to or lower than BadBreakWeight should not be considered for a break. A value greater than or equal to ForcedBreakWeight should be broken.

This is implemented to provide the default behavior of returning BadBreakWeight unless the length is greater than the length of the view in which case the entire view represents the fragment. Unless a view has been written to support breaking behavior, it is not attractive to try and break the view. An example of a view that does support breaking is LabelView. An example of a view that uses break weight is ParagraphView.

**Parameters:**axis - may be either View.X\_AXIS or View.Y\_AXISpos - the potential location of the start of the broken view >= 0. This may be useful for calculating tab positionslen - specifies the relative length from *pos* where a potential break is desired >= 0 **Returns:**the weight, which should be a value between ForcedBreakWeight and BadBreakWeight**See Also:**[LabelView](http://docs.google.com/javax/swing/text/LabelView.html), [ParagraphView](http://docs.google.com/javax/swing/text/ParagraphView.html), [BadBreakWeight](http://docs.google.com/javax/swing/text/View.html#BadBreakWeight), [GoodBreakWeight](http://docs.google.com/javax/swing/text/View.html#GoodBreakWeight), [ExcellentBreakWeight](http://docs.google.com/javax/swing/text/View.html#ExcellentBreakWeight), [ForcedBreakWeight](http://docs.google.com/javax/swing/text/View.html#ForcedBreakWeight)

### getResizeWeight

public int **getResizeWeight**(int axis)

Determines the resizability of the view along the given axis. A value of 0 or less is not resizable.

**Parameters:**axis - may be either View.X\_AXIS or View.Y\_AXIS **Returns:**the weight

### setSize

public void **setSize**(float width,  
 float height)

Sets the size of the view. This should cause layout of the view along the given axis, if it has any layout duties.

**Parameters:**width - the width >= 0height - the height >= 0

### getContainer

public [Container](http://docs.google.com/java/awt/Container.html) **getContainer**()

Fetches the container hosting the view. This is useful for things like scheduling a repaint, finding out the host components font, etc. The default implementation of this is to forward the query to the parent view.

**Returns:**the container, null if none

### getViewFactory

public [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) **getViewFactory**()

Fetches the ViewFactory implementation that is feeding the view hierarchy. Normally the views are given this as an argument to updates from the model when they are most likely to need the factory, but this method serves to provide it at other times.

**Returns:**the factory, null if none

### getToolTipText

public [String](http://docs.google.com/java/lang/String.html) **getToolTipText**(float x,  
 float y,  
 [Shape](http://docs.google.com/java/awt/Shape.html) allocation)

Returns the tooltip text at the specified location. The default implementation returns the value from the child View identified by the passed in location.

**Since:** 1.4 **See Also:**[JTextComponent.getToolTipText(java.awt.event.MouseEvent)](http://docs.google.com/javax/swing/text/JTextComponent.html#getToolTipText(java.awt.event.MouseEvent))

### getViewIndex

public int **getViewIndex**(float x,  
 float y,  
 [Shape](http://docs.google.com/java/awt/Shape.html) allocation)

Returns the child view index representing the given position in the view. This iterates over all the children returning the first with a bounds that contains x, y.

**Parameters:**x - the x coordinatey - the y coordinateallocation - current allocation of the View. **Returns:**index of the view representing the given location, or -1 if no view represents that position**Since:** 1.4

### updateChildren

protected boolean **updateChildren**([DocumentEvent.ElementChange](http://docs.google.com/javax/swing/event/DocumentEvent.ElementChange.html) ec,  
 [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e,  
 [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)

Updates the child views in response to receiving notification that the model changed, and there is change record for the element this view is responsible for. This is implemented to assume the child views are directly responsible for the child elements of the element this view represents. The ViewFactory is used to create child views for each element specified as added in the ElementChange, starting at the index specified in the given ElementChange. The number of child views representing the removed elements specified are removed.

**Parameters:**ec - the change information for the element this view is responsible for. This should not be null if this method gets callede - the change information from the associated documentf - the factory to use to build child views **Returns:**whether or not the child views represent the child elements of the element this view is responsible for. Some views create children that represent a portion of the element they are responsible for, and should return false. This information is used to determine if views in the range of the added elements should be forwarded to or not**Since:** 1.3 **See Also:**[insertUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#insertUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory)), [removeUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#removeUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory)), [changedUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#changedUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))

### forwardUpdate

protected void **forwardUpdate**([DocumentEvent.ElementChange](http://docs.google.com/javax/swing/event/DocumentEvent.ElementChange.html) ec,  
 [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)

Forwards the given DocumentEvent to the child views that need to be notified of the change to the model. If there were changes to the element this view is responsible for, that should be considered when forwarding (i.e. new child views should not get notified).

**Parameters:**ec - changes to the element this view is responsible for (may be null if there were no changes).e - the change information from the associated documenta - the current allocation of the viewf - the factory to use to rebuild if the view has children**Since:** 1.3 **See Also:**[insertUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#insertUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory)), [removeUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#removeUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory)), [changedUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#changedUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))

### forwardUpdateToView

protected void **forwardUpdateToView**([View](http://docs.google.com/javax/swing/text/View.html) v,  
 [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a,  
 [ViewFactory](http://docs.google.com/javax/swing/text/ViewFactory.html) f)

Forwards the DocumentEvent to the give child view. This simply messages the view with a call to insertUpdate, removeUpdate, or changedUpdate depending upon the type of the event. This is called by [forwardUpdate](#2iq8gzs) to forward the event to children that need it.

**Parameters:**v - the child view to forward the event toe - the change information from the associated documenta - the current allocation of the viewf - the factory to use to rebuild if the view has children**Since:** 1.3 **See Also:**[forwardUpdate(javax.swing.event.DocumentEvent.ElementChange, javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#forwardUpdate(javax.swing.event.DocumentEvent.ElementChange,%20javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))

### updateLayout

protected void **updateLayout**([DocumentEvent.ElementChange](http://docs.google.com/javax/swing/event/DocumentEvent.ElementChange.html) ec,  
 [DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html) e,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a)

Updates the layout in response to receiving notification of change from the model. This is implemented to call preferenceChanged to reschedule a new layout if the ElementChange record is not null.

**Parameters:**ec - changes to the element this view is responsible for (may be null if there were no changes)e - the change information from the associated documenta - the current allocation of the view**Since:** 1.3 **See Also:**[insertUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#insertUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory)), [removeUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#removeUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory)), [changedUpdate(javax.swing.event.DocumentEvent, java.awt.Shape, javax.swing.text.ViewFactory)](http://docs.google.com/javax/swing/text/View.html#changedUpdate(javax.swing.event.DocumentEvent,%20java.awt.Shape,%20javax.swing.text.ViewFactory))

### modelToView

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public [Shape](http://docs.google.com/java/awt/Shape.html) **modelToView**(int pos,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

**Deprecated.**

Provides a mapping from the document model coordinate space to the coordinate space of the view mapped to it. This is implemented to default the bias to Position.Bias.Forward which was previously implied.

**Parameters:**pos - the position to convert >= 0a - the allocated region in which to render **Returns:**the bounding box of the given position is returned **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) - if the given position does not represent a valid location in the associated document**See Also:**[modelToView(int, java.awt.Shape, javax.swing.text.Position.Bias)](http://docs.google.com/javax/swing/text/View.html#modelToView(int,%20java.awt.Shape,%20javax.swing.text.Position.Bias))

### viewToModel

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **viewToModel**(float x,  
 float y,  
 [Shape](http://docs.google.com/java/awt/Shape.html) a)

**Deprecated.**

Provides a mapping from the view coordinate space to the logical coordinate space of the model.

**Parameters:**x - the X coordinate >= 0y - the Y coordinate >= 0a - the allocated region in which to render **Returns:**the location within the model that best represents the given point in the view >= 0**See Also:**[viewToModel(float, float, java.awt.Shape, javax.swing.text.Position.Bias[])](http://docs.google.com/javax/swing/text/View.html#viewToModel(float,%20float,%20java.awt.Shape,%20javax.swing.text.Position.Bias%5B%5D))

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/View.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/text/Utilities.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/text/ViewFactory.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/text/View.html)    [**NO FRAMES**](http://docs.google.com/View.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: [FIELD](#4d34og8) | [CONSTR](#1ksv4uv) | [METHOD](#2jxsxqh) |

[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).