| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Document.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/DefaultTextUI.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/text/DocumentFilter.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/text/Document.html)    [**NO FRAMES**](http://docs.google.com/Document.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | CONSTR | [METHOD](#2et92p0) | DETAIL: [FIELD](#tyjcwt) | CONSTR | [METHOD](#4d34og8) |

## **javax.swing.text**

Interface Document

**All Known Subinterfaces:** [StyledDocument](http://docs.google.com/javax/swing/text/StyledDocument.html) **All Known Implementing Classes:** [AbstractDocument](http://docs.google.com/javax/swing/text/AbstractDocument.html), [DefaultStyledDocument](http://docs.google.com/javax/swing/text/DefaultStyledDocument.html), [HTMLDocument](http://docs.google.com/javax/swing/text/html/HTMLDocument.html), [PlainDocument](http://docs.google.com/javax/swing/text/PlainDocument.html)

public interface **Document**

The Document is a container for text that serves as the model for swing text components. The goal for this interface is to scale from very simple needs (a plain text textfield) to complex needs (an HTML or XML document, for example).

**Content**

At the simplest level, text can be modeled as a linear sequence of characters. To support internationalization, the Swing text model uses [unicode](http://www.unicode.org/) characters. The sequence of characters displayed in a text component is generally referred to as the component's *content*.

To refer to locations within the sequence, the coordinates used are the location between two characters. As the diagram below shows, a location in a text document can be referred to as a position, or an offset. This position is zero-based.



In the example, if the content of a document is the sequence "The quick brown fox," as shown in the preceding diagram, the location just before the word "The" is 0, and the location after the word "The" and before the whitespace that follows it is 3. The entire sequence of characters in the sequence "The" is called a *range*.

The following methods give access to the character data that makes up the content.

* [getLength()](#2s8eyo1)
* [getText(int, int)](#z337ya)
* [getText(int, int, Segment)](#3j2qqm3)

**Structure**

Text is rarely represented simply as featureless content. Rather, text typically has some sort of structure associated with it. Exactly what structure is modeled is up to a particular Document implementation. It might be as simple as no structure (i.e. a simple text field), or it might be something like diagram below.



The unit of structure (i.e. a node of the tree) is referred to by the [Element](http://docs.google.com/Element.html) interface. Each Element can be tagged with a set of attributes. These attributes (name/value pairs) are defined by the [AttributeSet](http://docs.google.com/AttributeSet.html) interface.

The following methods give access to the document structure.

* [getDefaultRootElement](#3whwml4)
* [getRootElements](#1ci93xb)

**Mutations**

All documents need to be able to add and remove simple text. Typically, text is inserted and removed via gestures from a keyboard or a mouse. What effect the insertion or removal has upon the document structure is entirely up to the implementation of the document.

The following methods are related to mutation of the document content:

* [insertString(int, String, AttributeSet)](#2jxsxqh)
* [remove(int, int)](#44sinio)
* [createPosition(int)](#2xcytpi)

**Notification**

Mutations to the Document must be communicated to interested observers. The notification of change follows the event model guidelines that are specified for JavaBeans. In the JavaBeans event model, once an event notification is dispatched, all listeners must be notified before any further mutations occur to the source of the event. Further, order of delivery is not guaranteed.

Notification is provided as two separate events, [DocumentEvent](http://docs.google.com/event/DocumentEvent.html), and [UndoableEditEvent](http://docs.google.com/event/UndoableEditEvent.html). If a mutation is made to a Document through its api, a DocumentEvent will be sent to all of the registered DocumentListeners. If the Document implementation supports undo/redo capabilities, an UndoableEditEvent will be sent to all of the registered UndoableEditListeners. If an undoable edit is undone, a DocumentEvent should be fired from the Document to indicate it has changed again. In this case however, there should be no UndoableEditEvent generated since that edit is actually the source of the change rather than a mutation to the Document made through its api.



Referring to the above diagram, suppose that the component shown on the left mutates the document object represented by the blue rectangle. The document responds by dispatching a DocumentEvent to both component views and sends an UndoableEditEvent to the listening logic, which maintains a history buffer.

Now suppose that the component shown on the right mutates the same document. Again, the document dispatches a DocumentEvent to both component views and sends an UndoableEditEvent to the listening logic that is maintaining the history buffer.

If the history buffer is then rolled back (i.e. the last UndoableEdit undone), a DocumentEvent is sent to both views, causing both of them to reflect the undone mutation to the document (that is, the removal of the right component's mutation). If the history buffer again rolls back another change, another DocumentEvent is sent to both views, causing them to reflect the undone mutation to the document -- that is, the removal of the left component's mutation.

The methods related to observing mutations to the document are:

* [addDocumentListener(DocumentListener)](#17dp8vu)
* [removeDocumentListener(DocumentListener)](#3rdcrjn)
* [addUndoableEditListener(UndoableEditListener)](#26in1rg)
* [removeUndoableEditListener(UndoableEditListener)](#lnxbz9)

**Properties**

Document implementations will generally have some set of properties associated with them at runtime. Two well known properties are the [StreamDescriptionProperty](#3dy6vkm), which can be used to describe where the Document came from, and the [TitleProperty](#1t3h5sf), which can be used to name the Document. The methods related to the properties are:

* [getProperty(Object)](#35nkun2)
* [putProperty(Object, Object)](#1ksv4uv)

For more information on the Document class, see [The Swing Connection](http://java.sun.com/products/jfc/tsc) and most particularly the article,  [The Element Interface](http://java.sun.com/products/jfc/tsc/articles/text/element_interface).

**See Also:**[DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html), [DocumentListener](http://docs.google.com/javax/swing/event/DocumentListener.html), [UndoableEditEvent](http://docs.google.com/javax/swing/event/UndoableEditEvent.html), [UndoableEditListener](http://docs.google.com/javax/swing/event/UndoableEditListener.html), [Element](http://docs.google.com/javax/swing/text/Element.html), [Position](http://docs.google.com/javax/swing/text/Position.html), [AttributeSet](http://docs.google.com/javax/swing/text/AttributeSet.html)

| **Field Summary** | |
| --- | --- |
| static [String](http://docs.google.com/java/lang/String.html) | [**StreamDescriptionProperty**](http://docs.google.com/javax/swing/text/Document.html#StreamDescriptionProperty)            The property name for the description of the stream used to initialize the document. |
| static [String](http://docs.google.com/java/lang/String.html) | [**TitleProperty**](http://docs.google.com/javax/swing/text/Document.html#TitleProperty)            The property name for the title of the document, if there is one. |

| **Method Summary** | |
| --- | --- |
| void | [**addDocumentListener**](http://docs.google.com/javax/swing/text/Document.html#addDocumentListener(javax.swing.event.DocumentListener))([DocumentListener](http://docs.google.com/javax/swing/event/DocumentListener.html) listener)            Registers the given observer to begin receiving notifications when changes are made to the document. |
| void | [**addUndoableEditListener**](http://docs.google.com/javax/swing/text/Document.html#addUndoableEditListener(javax.swing.event.UndoableEditListener))([UndoableEditListener](http://docs.google.com/javax/swing/event/UndoableEditListener.html) listener)            Registers the given observer to begin receiving notifications when undoable edits are made to the document. |
| [Position](http://docs.google.com/javax/swing/text/Position.html) | [**createPosition**](http://docs.google.com/javax/swing/text/Document.html#createPosition(int))(int offs)            This method allows an application to mark a place in a sequence of character content. |
| [Element](http://docs.google.com/javax/swing/text/Element.html) | [**getDefaultRootElement**](http://docs.google.com/javax/swing/text/Document.html#getDefaultRootElement())()            Returns the root element that views should be based upon, unless some other mechanism for assigning views to element structures is provided. |
| [Position](http://docs.google.com/javax/swing/text/Position.html) | [**getEndPosition**](http://docs.google.com/javax/swing/text/Document.html#getEndPosition())()            Returns a position that represents the end of the document. |
| int | [**getLength**](http://docs.google.com/javax/swing/text/Document.html#getLength())()            Returns number of characters of content currently in the document. |
| [Object](http://docs.google.com/java/lang/Object.html) | [**getProperty**](http://docs.google.com/javax/swing/text/Document.html#getProperty(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) key)            Gets the properties associated with the document. |
| [Element](http://docs.google.com/javax/swing/text/Element.html)[] | [**getRootElements**](http://docs.google.com/javax/swing/text/Document.html#getRootElements())()            Returns all of the root elements that are defined. |
| [Position](http://docs.google.com/javax/swing/text/Position.html) | [**getStartPosition**](http://docs.google.com/javax/swing/text/Document.html#getStartPosition())()            Returns a position that represents the start of the document. |
| [String](http://docs.google.com/java/lang/String.html) | [**getText**](http://docs.google.com/javax/swing/text/Document.html#getText(int,%20int))(int offset, int length)            Fetches the text contained within the given portion of the document. |
| void | [**getText**](http://docs.google.com/javax/swing/text/Document.html#getText(int,%20int,%20javax.swing.text.Segment))(int offset, int length, [Segment](http://docs.google.com/javax/swing/text/Segment.html) txt)            Fetches the text contained within the given portion of the document. |
| void | [**insertString**](http://docs.google.com/javax/swing/text/Document.html#insertString(int,%20java.lang.String,%20javax.swing.text.AttributeSet))(int offset, [String](http://docs.google.com/java/lang/String.html) str, [AttributeSet](http://docs.google.com/javax/swing/text/AttributeSet.html) a)            Inserts a string of content. |
| void | [**putProperty**](http://docs.google.com/javax/swing/text/Document.html#putProperty(java.lang.Object,%20java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) key, [Object](http://docs.google.com/java/lang/Object.html) value)            Associates a property with the document. |
| void | [**remove**](http://docs.google.com/javax/swing/text/Document.html#remove(int,%20int))(int offs, int len)            Removes a portion of the content of the document. |
| void | [**removeDocumentListener**](http://docs.google.com/javax/swing/text/Document.html#removeDocumentListener(javax.swing.event.DocumentListener))([DocumentListener](http://docs.google.com/javax/swing/event/DocumentListener.html) listener)            Unregisters the given observer from the notification list so it will no longer receive change updates. |
| void | [**removeUndoableEditListener**](http://docs.google.com/javax/swing/text/Document.html#removeUndoableEditListener(javax.swing.event.UndoableEditListener))([UndoableEditListener](http://docs.google.com/javax/swing/event/UndoableEditListener.html) listener)            Unregisters the given observer from the notification list so it will no longer receive updates. |
| void | [**render**](http://docs.google.com/javax/swing/text/Document.html#render(java.lang.Runnable))([Runnable](http://docs.google.com/java/lang/Runnable.html) r)            Allows the model to be safely rendered in the presence of concurrency, if the model supports being updated asynchronously. |

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

### StreamDescriptionProperty

static final [String](http://docs.google.com/java/lang/String.html) **StreamDescriptionProperty**

The property name for the description of the stream used to initialize the document. This should be used if the document was initialized from a stream and anything is known about the stream.

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

### TitleProperty

static final [String](http://docs.google.com/java/lang/String.html) **TitleProperty**

The property name for the title of the document, if there is one.

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

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

### getLength

int **getLength**()

Returns number of characters of content currently in the document.

**Returns:**number of characters >= 0

### addDocumentListener

void **addDocumentListener**([DocumentListener](http://docs.google.com/javax/swing/event/DocumentListener.html) listener)

Registers the given observer to begin receiving notifications when changes are made to the document.

**Parameters:**listener - the observer to register**See Also:**[removeDocumentListener(javax.swing.event.DocumentListener)](http://docs.google.com/javax/swing/text/Document.html#removeDocumentListener(javax.swing.event.DocumentListener))

### removeDocumentListener

void **removeDocumentListener**([DocumentListener](http://docs.google.com/javax/swing/event/DocumentListener.html) listener)

Unregisters the given observer from the notification list so it will no longer receive change updates.

**Parameters:**listener - the observer to register**See Also:**[addDocumentListener(javax.swing.event.DocumentListener)](http://docs.google.com/javax/swing/text/Document.html#addDocumentListener(javax.swing.event.DocumentListener))

### addUndoableEditListener

void **addUndoableEditListener**([UndoableEditListener](http://docs.google.com/javax/swing/event/UndoableEditListener.html) listener)

Registers the given observer to begin receiving notifications when undoable edits are made to the document.

**Parameters:**listener - the observer to register**See Also:**[UndoableEditEvent](http://docs.google.com/javax/swing/event/UndoableEditEvent.html)

### removeUndoableEditListener

void **removeUndoableEditListener**([UndoableEditListener](http://docs.google.com/javax/swing/event/UndoableEditListener.html) listener)

Unregisters the given observer from the notification list so it will no longer receive updates.

**Parameters:**listener - the observer to register**See Also:**[UndoableEditEvent](http://docs.google.com/javax/swing/event/UndoableEditEvent.html)

### getProperty

[Object](http://docs.google.com/java/lang/Object.html) **getProperty**([Object](http://docs.google.com/java/lang/Object.html) key)

Gets the properties associated with the document.

**Parameters:**key - a non-null property key **Returns:**the properties**See Also:**[putProperty(Object, Object)](http://docs.google.com/javax/swing/text/Document.html#putProperty(java.lang.Object,%20java.lang.Object))

### putProperty

void **putProperty**([Object](http://docs.google.com/java/lang/Object.html) key,  
 [Object](http://docs.google.com/java/lang/Object.html) value)

Associates a property with the document. Two standard property keys provided are:  [StreamDescriptionProperty](#3dy6vkm) and [TitleProperty](#1t3h5sf). Other properties, such as author, may also be defined.

**Parameters:**key - the non-null property keyvalue - the property value**See Also:**[getProperty(Object)](http://docs.google.com/javax/swing/text/Document.html#getProperty(java.lang.Object))

### remove

void **remove**(int offs,  
 int len)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

Removes a portion of the content of the document. This will cause a DocumentEvent of type DocumentEvent.EventType.REMOVE to be sent to the registered DocumentListeners, unless an exception is thrown. The notification will be sent to the listeners by calling the removeUpdate method on the DocumentListeners.

To ensure reasonable behavior in the face of concurrency, the event is dispatched after the mutation has occurred. This means that by the time a notification of removal is dispatched, the document has already been updated and any marks created by createPosition have already changed. For a removal, the end of the removal range is collapsed down to the start of the range, and any marks in the removal range are collapsed down to the start of the range.



If the Document structure changed as result of the removal, the details of what Elements were inserted and removed in response to the change will also be contained in the generated DocumentEvent. It is up to the implementation of a Document to decide how the structure should change in response to a remove.

If the Document supports undo/redo, an UndoableEditEvent will also be generated.

**Parameters:**offs - the offset from the beginning >= 0len - the number of characters to remove >= 0 **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) - some portion of the removal range was not a valid part of the document. The location in the exception is the first bad position encountered.**See Also:**[DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html), [DocumentListener](http://docs.google.com/javax/swing/event/DocumentListener.html), [UndoableEditEvent](http://docs.google.com/javax/swing/event/UndoableEditEvent.html), [UndoableEditListener](http://docs.google.com/javax/swing/event/UndoableEditListener.html)

### insertString

void **insertString**(int offset,  
 [String](http://docs.google.com/java/lang/String.html) str,  
 [AttributeSet](http://docs.google.com/javax/swing/text/AttributeSet.html) a)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

Inserts a string of content. This will cause a DocumentEvent of type DocumentEvent.EventType.INSERT to be sent to the registered DocumentListers, unless an exception is thrown. The DocumentEvent will be delivered by calling the insertUpdate method on the DocumentListener. The offset and length of the generated DocumentEvent will indicate what change was actually made to the Document.



If the Document structure changed as result of the insertion, the details of what Elements were inserted and removed in response to the change will also be contained in the generated DocumentEvent. It is up to the implementation of a Document to decide how the structure should change in response to an insertion.

If the Document supports undo/redo, an UndoableEditEvent will also be generated.

**Parameters:**offset - the offset into the document to insert the content >= 0. All positions that track change at or after the given location will move.str - the string to inserta - the attributes to associate with the inserted content. This may be null if there are no attributes. **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) - the given insert position is not a valid position within the document**See Also:**[DocumentEvent](http://docs.google.com/javax/swing/event/DocumentEvent.html), [DocumentListener](http://docs.google.com/javax/swing/event/DocumentListener.html), [UndoableEditEvent](http://docs.google.com/javax/swing/event/UndoableEditEvent.html), [UndoableEditListener](http://docs.google.com/javax/swing/event/UndoableEditListener.html)

### getText

[String](http://docs.google.com/java/lang/String.html) **getText**(int offset,  
 int length)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

Fetches the text contained within the given portion of the document.

**Parameters:**offset - the offset into the document representing the desired start of the text >= 0length - the length of the desired string >= 0 **Returns:**the text, in a String of length >= 0 **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) - some portion of the given range was not a valid part of the document. The location in the exception is the first bad position encountered.

### getText

void **getText**(int offset,  
 int length,  
 [Segment](http://docs.google.com/javax/swing/text/Segment.html) txt)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

Fetches the text contained within the given portion of the document.

If the partialReturn property on the txt parameter is false, the data returned in the Segment will be the entire length requested and may or may not be a copy depending upon how the data was stored. If the partialReturn property is true, only the amount of text that can be returned without creating a copy is returned. Using partial returns will give better performance for situations where large parts of the document are being scanned. The following is an example of using the partial return to access the entire document:

  int nleft = doc.getDocumentLength();  
   Segment text = new Segment();  
   int offs = 0;  
   text.setPartialReturn(true);   
   while (nleft > 0) {  
   doc.getText(offs, nleft, text);  
   // do someting with text  
   nleft -= text.count;  
   offs += text.count;  
   }

**Parameters:**offset - the offset into the document representing the desired start of the text >= 0length - the length of the desired string >= 0txt - the Segment object to return the text in **Throws:** [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html) - Some portion of the given range was not a valid part of the document. The location in the exception is the first bad position encountered.

### getStartPosition

[Position](http://docs.google.com/javax/swing/text/Position.html) **getStartPosition**()

Returns a position that represents the start of the document. The position returned can be counted on to track change and stay located at the beginning of the document.

**Returns:**the position

### getEndPosition

[Position](http://docs.google.com/javax/swing/text/Position.html) **getEndPosition**()

Returns a position that represents the end of the document. The position returned can be counted on to track change and stay located at the end of the document.

**Returns:**the position

### createPosition

[Position](http://docs.google.com/javax/swing/text/Position.html) **createPosition**(int offs)  
 throws [BadLocationException](http://docs.google.com/javax/swing/text/BadLocationException.html)

This method allows an application to mark a place in a sequence of character content. This mark can then be used to tracks change as insertions and removals are made in the content. The policy is that insertions always occur prior to the current position (the most common case) unless the insertion location is zero, in which case the insertion is forced to a position that follows the original position.

**Parameters:**offs - the offset from the start of the document >= 0 **Returns:**the position **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

### getRootElements

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

Returns all of the root elements that are defined.

Typically there will be only one document structure, but the interface supports building an arbitrary number of structural projections over the text data. The document can have multiple root elements to support multiple document structures. Some examples might be:

* Text direction.
* Lexical token streams.
* Parse trees.
* Conversions to formats other than the native format.
* Modification specifications.
* Annotations.

**Returns:**the root element

### getDefaultRootElement

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

Returns the root element that views should be based upon, unless some other mechanism for assigning views to element structures is provided.

**Returns:**the root element

### render

void **render**([Runnable](http://docs.google.com/java/lang/Runnable.html) r)

Allows the model to be safely rendered in the presence of concurrency, if the model supports being updated asynchronously. The given runnable will be executed in a way that allows it to safely read the model with no changes while the runnable is being executed. The runnable itself may *not* make any mutations.

**Parameters:**r - a Runnable used to render the model

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Document.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/DefaultTextUI.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/text/DocumentFilter.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/text/Document.html)    [**NO FRAMES**](http://docs.google.com/Document.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | CONSTR | [METHOD](#2et92p0) | DETAIL: [FIELD](#tyjcwt) | CONSTR | [METHOD](#4d34og8) |

[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).