

A Java™ API demo
documentation
generated with DocFlex/Javadoc
v1.6.0

Contents

Overview	3
Package Summary	3
All Classes Summary	3
java.lang	4
Enum	5
java.util	9
EnumSet	10
javax.swing.text	16
Document	17
JTextComponent	26
JTextComponent.AccessibleJTextComponent	55
JTextComponent.DropLocation	68
JTextComponent.KeyBinding	70

Overview

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

Package Summary		Page
java.lang	Provides classes that are fundamental to the design of the Java programming language.	3
java.util	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).	9
javax.swing.text	Provides classes and interfaces that deal with editable and noneditable text components.	16

All Classes Summary		Page
Document	The <code>Document</code> is a container for text that serves as the model for swing text components.	16
Enum<E extends Enum<E>>	This is the common base class of all Java language enumeration types.	4
EnumSet<E extends Enum<E>>	A specialized <code>Set</code> implementation for use with enum types.	9
JTextComponent	<code>JTextComponent</code> is the base class for swing text components.	26
JTextComponent.DropLocation	Represents a drop location for <code>JTextComponents</code> .	68
JTextComponent.KeyBinding	Binding record for creating key bindings.	70

Package java.lang

Provides classes that are fundamental to the design of the Java programming language. The most important classes are `Object`, which is the root of the class hierarchy, and `Class`, instances of which represent classes at run time.

Frequently it is necessary to represent a value of primitive type as if it were an object. The wrapper classes `Boolean`, `Character`, `Integer`, `Long`, `Float`, and `Double` serve this purpose. An object of type `Double`, for example, contains a field whose type is `double`, representing that value in such a way that a reference to it can be stored in a variable of reference type. These classes also provide a number of methods for converting among primitive values, as well as supporting such standard methods as `equals` and `hashCode`. The `Void` class is a non-instantiable class that holds a reference to a `Class` object representing the primitive type `void`.

The class `Math` provides commonly used mathematical functions such as sine, cosine, and square root. The classes `String` and `StringBuffer` similarly provide commonly used operations on character strings.

Classes `ClassLoader`, `Process`, `Runtime`, `SecurityManager`, and `System` provide "system operations" that manage the dynamic loading of classes, creation of external processes, host environment inquiries such as the time of day, and enforcement of security policies.

Class `Throwable` encompasses objects that may be thrown by the `throw` statement (§14.16). Subclasses of `Throwable` represent errors and exceptions.

Package Specification

Character Encodings

The specification of the `java.nio.charset.Charset` class describes the naming conventions for character encodings as well as the set of standard encodings that must be supported by every implementation of the Java platform.

Since:

JDK1.0

Class Summary		Page
Enum<E extends Enum<E>>	This is the common base class of all Java language enumeration types.	4

java.lang Enum

```
java.lang.Object
└─ java.lang.Enum<E>
```

All Implemented Interfaces:

Comparable<E>, Serializable

```
public abstract class Enum<E extends Enum<E>>
extends Object
implements Comparable<E>, Serializable
```

This is the common base class of all Java language enumeration types.

Since:

1.5

Author:

Josh Bloch, Neal Gafter

Constructor Summary		Page
protected	Enum (String name, int ordinal) Sole constructor.	6

Method Summary		Page
protected Object	clone () Throws CloneNotSupportedException.	7
int	compareTo (E o) Compares this enum with the specified object for order.	7
boolean	equals (Object other) Returns true if the specified object is equal to this enum constant.	7
protected void	finalize () enum classes cannot have finalize methods.	8
Class<E>	getDeclaringClass () Returns the Class object corresponding to this enum constant's enum type.	7
int	hashCode () Returns a hash code for this enum constant.	7
String	name () Returns the name of this enum constant, exactly as declared in its enum declaration.	6
int	ordinal () Returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero).	6
String	toString () Returns the name of this enum constant, as contained in the declaration.	6
static <T extends Enum<T>> T	valueOf (Class<T> enumType, String name) Returns the enum constant of the specified enum type with the specified name.	8

Constructor Detail

Enum

```
protected Enum(String name,  
                int ordinal)
```

Sole constructor. Programmers cannot invoke this constructor. It is for use by code emitted by the compiler in response to enum type declarations.

Parameters:

`name` - - The name of this enum constant, which is the identifier used to declare it.

`ordinal` - - The ordinal of this enumeration constant (its position in the enum declaration, where the initial constant is assigned an ordinal of zero).

Method Detail

name

```
public final String name()
```

Returns the name of this enum constant, exactly as declared in its enum declaration. **Most programmers should use the `toString()` method in preference to this one, as the `toString` method may return a more user-friendly name.** This method is designed primarily for use in specialized situations where correctness depends on getting the exact name, which will not vary from release to release.

Returns:

the name of this enum constant

ordinal

```
public final int ordinal()
```

Returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero). Most programmers will have no use for this method. It is designed for use by sophisticated enum-based data structures, such as [EnumSet](#) and [EnumMap](#).

Returns:

the ordinal of this enumeration constant

toString

```
public String toString()
```

Returns the name of this enum constant, as contained in the declaration. This method may be overridden, though it typically isn't necessary or desirable. An enum type should override this method when a more "programmer-friendly" string form exists.

Overrides:

`toString` in class `Object`

Returns:

the name of this enum constant

equals

```
public final boolean equals(Object other)
```

Returns true if the specified object is equal to this enum constant.

Overrides:

equals in class Object

Parameters:

other - the object to be compared for equality with this object.

Returns:

true if the specified object is equal to this enum constant.

hashCode

```
public final int hashCode()
```

Returns a hash code for this enum constant.

Overrides:

hashCode in class Object

Returns:

a hash code for this enum constant.

clone

```
protected final Object clone()  
    throws CloneNotSupportedException
```

Throws CloneNotSupportedException. This guarantees that enums are never cloned, which is necessary to preserve their "singleton" status.

Overrides:

clone in class Object

Returns:

(never returns)

Throws:

CloneNotSupportedException

compareTo

```
public final int compareTo(E o)
```

Compares this enum with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object. Enum constants are only comparable to other enum constants of the same enum type. The natural order implemented by this method is the order in which the constants are declared.

Specified by:

compareTo in interface Comparable<T>

getDeclaringClass

```
public final Class<E> getDeclaringClass()
```

Returns the Class object corresponding to this enum constant's enum type. Two enum constants e1 and e2 are of the same enum type if and only if e1.getDeclaringClass() == e2.getDeclaringClass(). (The value returned by this method may differ from the one returned by the Object.getClass() method for enum constants with constant-specific class bodies.)

Returns:

the Class object corresponding to this enum constant's enum type

valueOf

```
public static <T extends Enum<T>> T valueOf(Class<T> enumType,  
                                             String name)
```

Returns the enum constant of the specified enum type with the specified name. The name must match exactly an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

enumType - the Class object of the enum type from which to return a constant
name - the name of the constant to return

Returns:

the enum constant of the specified enum type with the specified name

Throws:

IllegalArgumentException - if the specified enum type has no constant with the specified name, or the specified class object does not represent an enum type
NullPointerException - if enumType or name is null

Since:

1.5

finalize

```
protected final void finalize()
```

enum classes cannot have finalize methods.

Overrides:

finalize in class Object

Package java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Package Specification

- [Collections Framework Overview](#)
- [Collections Framework Annotated Outline](#)

Related Documentation

For overviews, tutorials, examples, guides, and tool documentation, please see:

- [Collections Framework Tutorial](#)
- [Collections Framework Design FAQ](#)

Since:

JDK1.0

Class Summary		Page
EnumSet<E extends Enum<E>>	A specialized Set implementation for use with enum types.	9

java.util

EnumSet

```
java.lang.Object
├─ java.util.AbstractCollection<E>
│   └─ java.util.AbstractSet<E>
│       └─ java.util.EnumSet<E>
```

All Implemented Interfaces:

Cloneable, Serializable, Set<E>, Collection<E>, Iterable<E>

```
public abstract class EnumSet<E> extends Enum<E>>
    extends AbstractSet<E>
    implements Cloneable, Serializable
```

A specialized Set implementation for use with enum types. All of the elements in an enum set must come from a single enum type that is specified, explicitly or implicitly, when the set is created. Enum sets are represented internally as bit vectors. This representation is extremely compact and efficient. The space and time performance of this class should be good enough to allow its use as a high-quality, typesafe alternative to traditional int-based "bit flags." Even bulk operations (such as `containsAll` and `retainAll`) should run very quickly if their argument is also an enum set.

The iterator returned by the `iterator` method traverses the elements in their *natural order* (the order in which the enum constants are declared). The returned iterator is *weakly consistent*: it will never throw `ConcurrentModificationException` and it may or may not show the effects of any modifications to the set that occur while the iteration is in progress.

Null elements are not permitted. Attempts to insert a null element will throw `NullPointerException`. Attempts to test for the presence of a null element or to remove one will, however, function properly.

Like most collection implementations, `EnumSet` is not synchronized. If multiple threads access an enum set concurrently, and at least one of the threads modifies the set, it should be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the enum set. If no such object exists, the set should be "wrapped" using the `Collections.synchronizedSet(Set)` method. This is best done at creation time, to prevent accidental unsynchronized access:

```
Set<MyEnum> s = Collections.synchronizedSet(EnumSet.noneOf(MyEnum.class));
```

Implementation note: All basic operations execute in constant time. They are likely (though not guaranteed) to be much faster than their `HashSet` counterparts. Even bulk operations execute in constant time if their argument is also an enum set.

This class is a member of the [Java Collections Framework](#).

Since:

1.5

Author:

Josh Bloch

See Also:

`EnumMap`

Method Summary		Page
<pre>static <E> extends Enum<E>> EnumSet<E></pre>	<pre>allOf(Class<E> elementType) Creates an enum set containing all of the elements in the specified element type.</pre>	11
<pre>EnumSet<E></pre>	<pre>clone() Returns a copy of this set.</pre>	15

<pre>static <E extends Enum<E>> EnumSet<E></pre>	complementOf (EnumSet<E> s) Creates an enum set with the same element type as the specified enum set, initially containing all the elements of this type that are <i>not</i> contained in the specified set.	12
<pre>static <E extends Enum<E>> EnumSet<E></pre>	copyOf (Collection<E> c) Creates an enum set initialized from the specified collection.	12
<pre>static <E extends Enum<E>> EnumSet<E></pre>	copyOf (EnumSet<E> s) Creates an enum set with the same element type as the specified enum set, initially containing the same elements (if any).	12
<pre>static <E extends Enum<E>> EnumSet<E></pre>	noneOf (Class<E> elementType) Creates an empty enum set with the specified element type.	11
<pre>static <E extends Enum<E>> EnumSet<E></pre>	of (E e) Creates an enum set initially containing the specified element.	12
<pre>static <E extends Enum<E>> EnumSet<E></pre>	of (E e1, E e2) Creates an enum set initially containing the specified elements.	13
<pre>static <E extends Enum<E>> EnumSet<E></pre>	of (E e1, E e2, E e3) Creates an enum set initially containing the specified elements.	13
<pre>static <E extends Enum<E>> EnumSet<E></pre>	of (E e1, E e2, E e3, E e4) Creates an enum set initially containing the specified elements.	13
<pre>static <E extends Enum<E>> EnumSet<E></pre>	of (E e1, E e2, E e3, E e4, E e5) Creates an enum set initially containing the specified elements.	14
<pre>static <E extends Enum<E>> EnumSet<E></pre>	of (E first, E... rest) Creates an enum set initially containing the specified elements.	14
<pre>static <E extends Enum<E>> EnumSet<E></pre>	range (E from, E to) Creates an enum set initially containing all of the elements in the range defined by the two specified endpoints.	14

Method Detail

noneOf

```
public static <E extends Enum<E>> EnumSet<E> noneOf(Class<E> elementType)
```

Creates an empty enum set with the specified element type.

Parameters:

elementType - the class object of the element type for this enum set

Throws:

NullPointerException - if elementType is null

allOf

```
public static <E extends Enum<E>> EnumSet<E> allOf(Class<E> elementType)
```

Creates an enum set containing all of the elements in the specified element type.

Parameters:

elementType - the class object of the element type for this enum set

Throws:

NullPointerException - if elementType is null

copyOf

```
public static <E extends Enum<E>> EnumSet<E> copyOf(EnumSet<E> s)
```

Creates an enum set with the same element type as the specified enum set, initially containing the same elements (if any).

Parameters:

s - the enum set from which to initialize this enum set

Throws:

NullPointerException - if s is null

copyOf

```
public static <E extends Enum<E>> EnumSet<E> copyOf(Collection<E> c)
```

Creates an enum set initialized from the specified collection. If the specified collection is an EnumSet instance, this static factory method behaves identically to `copyOf(EnumSet)`. Otherwise, the specified collection must contain at least one element (in order to determine the new enum set's element type).

Parameters:

c - the collection from which to initialize this enum set

Throws:

IllegalArgumentException - if c is not an EnumSet instance and contains no elements

NullPointerException - if c is null

complementOf

```
public static <E extends Enum<E>> EnumSet<E> complementOf(EnumSet<E> s)
```

Creates an enum set with the same element type as the specified enum set, initially containing all the elements of this type that are *not* contained in the specified set.

Parameters:

s - the enum set from whose complement to initialize this enum set

Throws:

NullPointerException - if s is null

of

```
public static <E extends Enum<E>> EnumSet<E> of(E e)
```

Creates an enum set initially containing the specified element. Overloadings of this method exist to initialize an enum set with one through five elements. A sixth overloading is provided that uses the varargs feature. This overloading may be used to create an enum set initially containing an arbitrary number of elements, but is likely to run slower than the overloadings that do not use varargs.

Parameters:

e - the element that this set is to contain initially

Returns:

an enum set initially containing the specified element

Throws:

NullPointerException - if e is null

of

```
public static <E extends Enum<E>> EnumSet<E> of(E e1,  
                                                E e2)
```

Creates an enum set initially containing the specified elements. Overloadings of this method exist to initialize an enum set with one through five elements. A sixth overloading is provided that uses the varargs feature. This overloading may be used to create an enum set initially containing an arbitrary number of elements, but is likely to run slower than the overloadings that do not use varargs.

Parameters:

- e1 - an element that this set is to contain initially
- e2 - another element that this set is to contain initially

Returns:

an enum set initially containing the specified elements

Throws:

`NullPointerException` - if any parameters are null

of

```
public static <E extends Enum<E>> EnumSet<E> of(E e1,  
                                                E e2,  
                                                E e3)
```

Creates an enum set initially containing the specified elements. Overloadings of this method exist to initialize an enum set with one through five elements. A sixth overloading is provided that uses the varargs feature. This overloading may be used to create an enum set initially containing an arbitrary number of elements, but is likely to run slower than the overloadings that do not use varargs.

Parameters:

- e1 - an element that this set is to contain initially
- e2 - another element that this set is to contain initially
- e3 - another element that this set is to contain initially

Returns:

an enum set initially containing the specified elements

Throws:

`NullPointerException` - if any parameters are null

of

```
public static <E extends Enum<E>> EnumSet<E> of(E e1,  
                                                E e2,  
                                                E e3,  
                                                E e4)
```

Creates an enum set initially containing the specified elements. Overloadings of this method exist to initialize an enum set with one through five elements. A sixth overloading is provided that uses the varargs feature. This overloading may be used to create an enum set initially containing an arbitrary number of elements, but is likely to run slower than the overloadings that do not use varargs.

Parameters:

- e1 - an element that this set is to contain initially
- e2 - another element that this set is to contain initially
- e3 - another element that this set is to contain initially
- e4 - another element that this set is to contain initially

Returns:

an enum set initially containing the specified elements

Throws:

NullPointerException - if any parameters are null

of

```
public static <E extends Enum<E>> EnumSet<E> of(E e1,
                                                E e2,
                                                E e3,
                                                E e4,
                                                E e5)
```

Creates an enum set initially containing the specified elements. Overloadings of this method exist to initialize an enum set with one through five elements. A sixth overloading is provided that uses the varargs feature. This overloading may be used to create an enum set initially containing an arbitrary number of elements, but is likely to run slower than the overloadings that do not use varargs.

Parameters:

- e1 - an element that this set is to contain initially
- e2 - another element that this set is to contain initially
- e3 - another element that this set is to contain initially
- e4 - another element that this set is to contain initially
- e5 - another element that this set is to contain initially

Returns:

an enum set initially containing the specified elements

Throws:

NullPointerException - if any parameters are null

of

```
public static <E extends Enum<E>> EnumSet<E> of(E first,
                                                E... rest)
```

Creates an enum set initially containing the specified elements. This factory, whose parameter list uses the varargs feature, may be used to create an enum set initially containing an arbitrary number of elements, but it is likely to run slower than the overloadings that do not use varargs.

Parameters:

- first - an element that the set is to contain initially
- rest - the remaining elements the set is to contain initially

Returns:

an enum set initially containing the specified elements

Throws:

NullPointerException - if any of the specified elements are null, or if rest is null

range

```
public static <E extends Enum<E>> EnumSet<E> range(E from,
                                                    E to)
```

Creates an enum set initially containing all of the elements in the range defined by the two specified endpoints. The returned set will contain the endpoints themselves, which may be identical but must not be out of order.

Parameters:

- from - the first element in the range
- to - the last element in the range

Returns:

an enum set initially containing all of the elements in the range defined by the two specified endpoints

Throws:

NullPointerException - if first or last are null

IllegalArgumentException - if `first.compareTo(last) > 0`

clone

```
public EnumSet<E> clone()
```

Returns a copy of this set.

Overrides:

clone in class Object

Returns:

a copy of this set

Package

javax.swing.text

Provides classes and interfaces that deal with editable and noneditable text components. Examples of text components are text fields and text areas, of which password fields and document editors are special instantiations. Features that are supported by this package include selection/highlighting, editing, style, and key mapping.

Note: Most of the Swing API is *not* thread safe. For details, see [Threads and Swing](#), a section in *The Java Tutorial*.

Related Documentation

For overviews, tutorials, examples, guides, and tool documentation, please see:

- [Using Text Components](#), a section in *The Java Tutorial*

Since:

1.2

Interface Summary		Page
Document	The <code>Document</code> is a container for text that serves as the model for swing text components.	16

Class Summary		Page
JTextComponent	<code>JTextComponent</code> is the base class for swing text components.	26
JTextComponent.DropLocation	Represents a drop location for <code>JTextComponents</code> .	68
JTextComponent.KeyBinding	Binding record for creating key bindings.	70

javafx.swing.text

Document

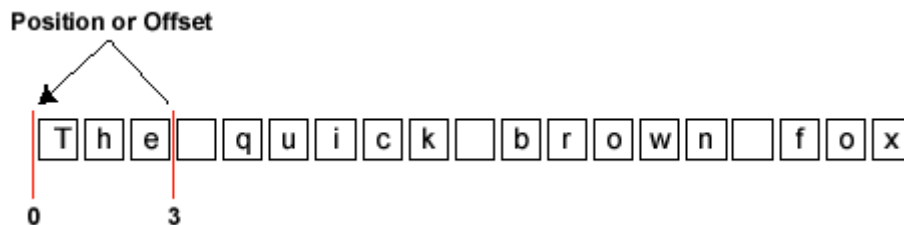
```
public interface Document
```

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](#) 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.



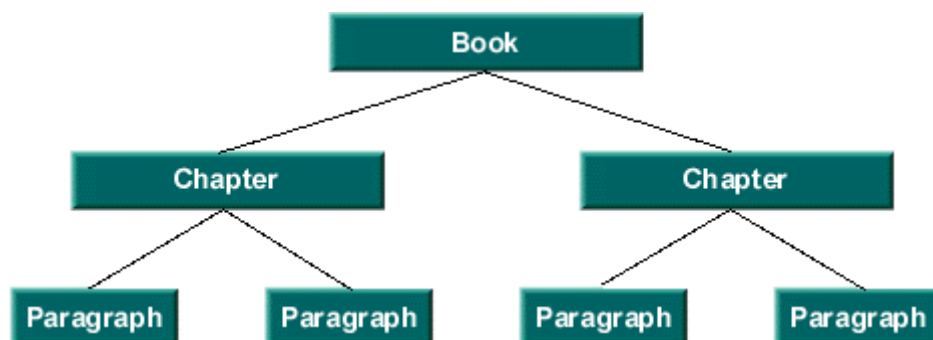
In the example, if the content of a document is the sequence "The quick brown fox," as shown in the preceding diagram, the location just before the word "The" is 0, and the location after the word "The" and before the whitespace that follows it is 3. The entire sequence of characters in the sequence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

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

- [getLength\(\)](#)
- [getText\(int, int\)](#)
- [getText\(int, int, Segment\)](#)

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](#) interface. Each `Element` can be tagged with a set of attributes. These attributes (name/value pairs) are defined by the [AttributeSet](#) interface.

The following methods give access to the document structure.

- [getDefaultRootElement](#)
- [getRootElements](#)

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.

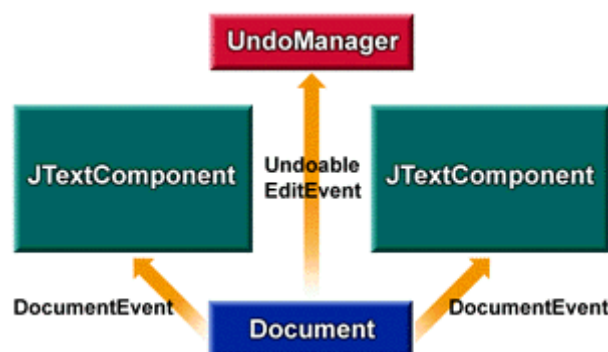
The following methods are related to mutation of the document content:

- [insertString\(int, String, AttributeSet\)](#)
- [remove\(int, int\)](#)
- [createPosition\(int\)](#)

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](#), and [UndoableEditEvent](#). 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\(DocumentListener\)](#)
- [removeDocumentListener\(DocumentListener\)](#)
- [addUndoableEditListener\(UndoableEditListener\)](#)
- [removeUndoableEditListener\(UndoableEditListener\)](#)

Properties

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

- [getProperty\(Object\)](#)
- [putProperty\(Object, Object\)](#)

For more information on the Document class, see [The Swing Connection](#) and most particularly the article, [The Element Interface](#).

Author:

Timothy Prinzing

See Also:

DocumentEvent, DocumentListener, UndoableEditEvent, UndoableEditListener, Element, Position, AttributeSet

Field Summary		Page
String	StreamDescriptionProperty The property name for the description of the stream used to initialize the document.	20
String	TitleProperty The property name for the title of the document, if there is one.	20

Method Summary		Page
void	addDocumentListener (DocumentListener listener) Registers the given observer to begin receiving notifications when changes are made to the document.	20
void	addUndoableEditListener (UndoableEditListener listener) Registers the given observer to begin receiving notifications when undoable edits are made to the document.	21
Position	createPosition (int offs) This method allows an application to mark a place in a sequence of character content.	24
Element	getDefaultRootElement () Returns the root element that views should be based upon, unless some other mechanism for assigning views to element structures is provided.	25
Position	getEndPosition () Returns a position that represents the end of the document.	24
int	getLength () Returns number of characters of content currently in the document.	20
Object	getProperty (Object key) Gets the properties associated with the document.	21
Element[]	getRootElements () Returns all of the root elements that are defined.	25
Position	getStartPosition () Returns a position that represents the start of the document.	24
String	getText (int offset, int length) Fetches the text contained within the given portion of the document.	23
void	getText (int offset, int length, Segment txt) Fetches the text contained within the given portion of the document.	23
void	insertString (int offset, String str, AttributeSet a) Inserts a string of content.	22

void	putProperty (Object key, Object value) Associates a property with the document.	21
void	remove (int offs, int len) Removes a portion of the content of the document.	22
void	removeDocumentListener (DocumentListener listener) Unregisters the given observer from the notification list so it will no longer receive change updates.	21
void	removeUndoableEditListener (UndoableEditListener listener) Unregisters the given observer from the notification list so it will no longer receive updates.	21
void	render (Runnable r) Allows the model to be safely rendered in the presence of concurrency, if the model supports being updated asynchronously.	25

Field Detail

StreamDescriptionProperty

```
public static final String 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.

TitleProperty

```
public static final String TitleProperty
```

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

Method Detail

getLength

```
int getLength()
```

Returns number of characters of content currently in the document.

Returns:

number of characters ≥ 0

addDocumentListener

```
void addDocumentListener(DocumentListener 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\(DocumentListener\)](#)

removeDocumentListener

void **removeDocumentListener**(DocumentListener 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\(DocumentListener\)](#)

addUndoableEditListener

void **addUndoableEditListener**(UndoableEditListener 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

removeUndoableEditListener

void **removeUndoableEditListener**(UndoableEditListener 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

getProperty

Object **getProperty**(Object key)

Gets the properties associated with the document.

Parameters:

key - a non-null property key

Returns:

the properties

See Also:

[putProperty\(Object, Object\)](#)

putProperty

void **putProperty**(Object key,
Object value)

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

Parameters:

key - the non-null property key
value - the property value

See Also:

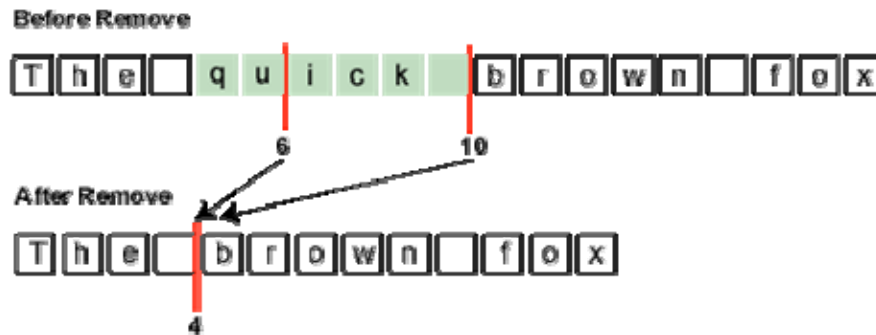
[getProperty\(Object\)](#)

remove

```
void remove(int offs,
             int len)
    throws BadLocationException
```

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 ≥ 0
len - the number of characters to remove ≥ 0

Throws:

BadLocationException
BadLocationException - 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:

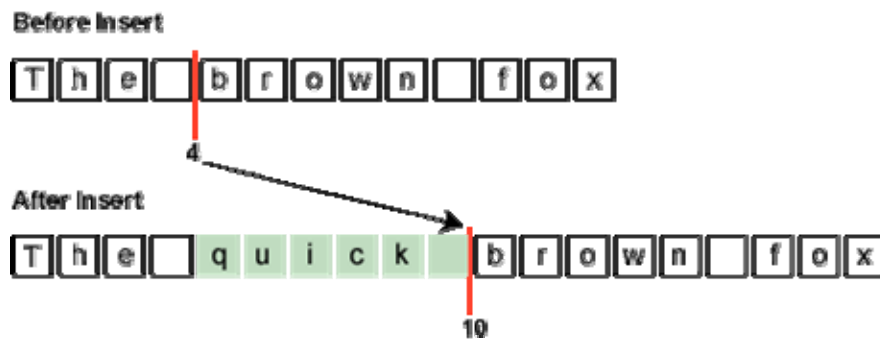
DocumentEvent, DocumentListener, UndoableEditEvent, UndoableEditListener

insertString

```
void insertString(int offset,
                  String str,
                  AttributeSet a)
    throws BadLocationException
```

Inserts a string of content. This will cause a DocumentEvent of type DocumentEvent.EventType.INSERT to be sent to the registered DocumentListeners, 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 insert
- a - the attributes to associate with the inserted content. This may be null if there are no attributes.

Throws:

- BadLocationException
- BadLocationException - the given insert position is not a valid position within the document

See Also:

DocumentEvent, DocumentListener, UndoableEditEvent, UndoableEditListener

getText

```
String getText(int offset,
               int length)
    throws BadLocationException
```

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 ≥ 0
- length - the length of the desired string ≥ 0

Returns:

the text, in a String of length ≥ 0

Throws:

- BadLocationException
- BadLocationException - 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 txt)
    throws BadLocationException
```

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 something with text
    nleft -= text.count;
    offs += text.count;
}
```

Parameters:

`offset` - the offset into the document representing the desired start of the text ≥ 0
`length` - the length of the desired string ≥ 0
`txt` - the `Segment` object to return the text in

Throws:

`BadLocationException`
`BadLocationException` - 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 `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 `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 `createPosition(int offs)`
throws `BadLocationException`

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

BadLocationException - if the given position does not represent a valid location in the associated document

getRootElements

Element[] **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 **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 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

javafx.swing.text

JTextComponent

```
java.lang.Object
├── java.awt.Component
│   ├── java.awt.Container
│   │   ├── javafx.swing.JComponent
│   │   └── javafx.swing.text.JTextComponent
```

All Implemented Interfaces:

Scrollable, Accessible, Serializable, ImageObserver, MenuContainer

```
public abstract class JTextComponent
extends JComponent
implements Scrollable, Accessible
```

`JTextComponent` is the base class for swing text components. It tries to be compatible with the `java.awt.TextComponent` class where it can reasonably do so. Also provided are other services for additional flexibility (beyond the pluggable UI and bean support). You can find information on how to use the functionality this class provides in [General Rules for Using Text Components](#), a section in *The Java Tutorial*.

Caret Changes

The caret is a pluggable object in swing text components. Notification of changes to the caret position and the selection are sent to implementations of the `CaretListener` interface that have been registered with the text component. The UI will install a default caret unless a customized caret has been set.

By default the caret tracks all the document changes performed on the Event Dispatching Thread and updates its position accordingly if an insertion occurs before or at the caret position or a removal occurs before the caret position. `DefaultCaret` tries to make itself visible which may lead to scrolling of a text component within `JScrollPane`.

The default caret behavior can be changed by the `DefaultCaret.setUpdatePolicy(int)` method.

Note: Non-editable text components also have a caret though it may not be painted.

Commands

Text components provide a number of commands that can be used to manipulate the component. This is essentially the way that the component expresses its capabilities. These are expressed in terms of the swing `Action` interface, using the `TextAction` implementation. The set of commands supported by the text component can be found with the `getActions()` method. These actions can be bound to key events, fired from buttons, etc.

Text Input

The text components support flexible and internationalized text input, using keymaps and the input method framework, while maintaining compatibility with the AWT listener model.

A `Keymap` lets an application bind key strokes to actions. In order to allow keymaps to be shared across multiple text components, they can use actions that extend `TextAction`. `TextAction` can determine which `JTextComponent` most recently has or had focus and therefore is the subject of the action (In the case that the `ActionEvent` sent to the action doesn't contain the target text component as its source).

The [input method framework](#) lets text components interact with input methods, separate software components that preprocess events to let users enter thousands of different characters using keyboards with far fewer keys. `JTextComponent` is an *active client* of the framework, so it implements the preferred user interface for interacting with input methods. As a consequence, some key events do not reach the text component because they are handled by an input method, and some text input reaches the text component as committed text within an `InputMethodEvent` instead of as a key event. The complete text input is the combination of the characters in `keyTyped` key events and committed text in input method events.

The AWT listener model lets applications attach event listeners to components in order to bind events to actions. Swing encourages the use of keymaps instead of listeners, but maintains compatibility with listeners by giving the listeners a chance to steal an event by consuming it.

Keyboard event and input method events are handled in the following stages, with each stage capable of consuming the event:

Stage	KeyEvent	InputMethodEvent
1.	input methods	(generated here)
2.	focus manager	
3.	registered key listeners	registered input method listeners
4.		input method handling in JTextComponent
5.	keymap handling using the current keymap	
6.	keyboard handling in JComponent (e.g. accelerators, component navigation, etc.)	

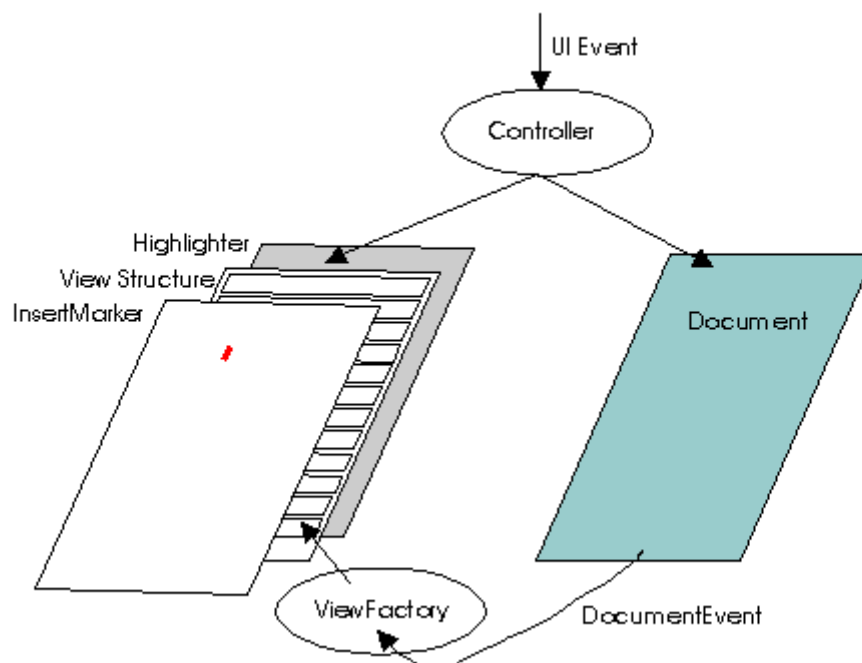
To maintain compatibility with applications that listen to key events but are not aware of input method events, the input method handling in stage 4 provides a compatibility mode for components that do not process input method events. For these components, the committed text is converted to keyTyped key events and processed in the key event pipeline starting at stage 3 instead of in the input method event pipeline.

By default the component will create a keymap (named **DEFAULT_KEYMAP**) that is shared by all JTextComponent instances as the default keymap. Typically a look-and-feel implementation will install a different keymap that resolves to the default keymap for those bindings not found in the different keymap. The minimal bindings include:

- inserting content into the editor for the printable keys.
- removing content with the backspace and del keys.
- caret movement forward and backward

Model/View Split

The text components have a model-view split. A text component pulls together the objects used to represent the model, view, and controller. The text document model may be shared by other views which act as observers of the model (e.g. a document may be shared by multiple components).



The model is defined by the [Document](#) interface. This is intended to provide a flexible text storage mechanism that tracks change during edits and can be extended to more sophisticated models. The model interfaces are meant to capture the capabilities of expression given by SGML, a system used to express a wide variety of content. Each modification to the document causes notification of the details of the change to be sent to all observers in the form of a `DocumentEvent` which allows the views to stay up to date with the model. This event is sent to observers that have implemented the `DocumentListener` interface and registered interest with the model being observed.

Location Information

The capability of determining the location of text in the view is provided. There are two methods, `modelToView(int)` and `viewToModel(Point)` for determining this information.

Undo/Redo support

Support for an edit history mechanism is provided to allow undo/redo operations. The text component does not itself provide the history buffer by default, but does provide the `UndoableEdit` records that can be used in conjunction with a history buffer to provide the undo/redo support. The support is provided by the Document model, which allows one to attach `UndoableEditListener` implementations.

Thread Safety

The swing text components provide some support of thread safe operations. Because of the high level of configurability of the text components, it is possible to circumvent the protection provided. The protection primarily comes from the model, so the documentation of `AbstractDocument` describes the assumptions of the protection provided. The methods that are safe to call asynchronously are marked with comments.

Newlines

For a discussion on how newlines are handled, see [DefaultEditorKit](#).

Printing support

Several `print` methods are provided for basic document printing. If more advanced printing is needed, use the `getPrintable(MessageFormat, MessageFormat)` method.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the `java.beans` package. Please see `XMLEncoder`.

Author:

Timothy Prinzing, Igor Kushnirskiy (printing support)

See Also:

[Document](#), `DocumentEvent`, `DocumentListener`, `Caret`, `CaretEvent`, `CaretListener`, `TextUI`, `View`, `ViewFactory`

Nested Class Summary		Page
class	JTextComponent.AccessibleJTextComponent This class implements accessibility support for the <code>JTextComponent</code> class.	55
static final class	JTextComponent.DropLocation Represents a drop location for <code>JTextComponents</code> .	68
static class	JTextComponent.KeyBinding Binding record for creating key bindings.	70

Field Summary		Page
static String	DEFAULT_KEYMAP The default keymap that will be shared by all <code>JTextComponent</code> instances unless they have had a different keymap set.	32
static String	FOCUS_ACCELERATOR_KEY The bound property name for the focus accelerator.	32

Constructor Summary		Page
<code>JTextComponent()</code> Creates a new JTextComponent.		32

Method Summary		Page
void	<code>addCaretListener(CaretListener listener)</code> Adds a caret listener for notification of any changes to the caret.	33
void	<code>addInputMethodListener(InputMethodListener l)</code>	54
static Keymap	<code>addKeymap(String nm, Keymap parent)</code> Adds a new keymap into the keymap hierarchy.	39
void	<code>copy()</code> Transfers the currently selected range in the associated text model to the system clipboard, leaving the contents in the text model.	43
void	<code>cut()</code> Transfers the currently selected range in the associated text model to the system clipboard, removing the contents from the model.	43
protected void	<code>fireCaretUpdate(CaretEvent e)</code> Notifies all listeners that have registered interest for notification on this event type.	34
AccessibleContext	<code>getAccessibleContext()</code> Gets the AccessibleContext associated with this JTextComponent.	53
Action[]	<code>getActions()</code> Fetches the command list for the editor.	35
Caret	<code>getCaret()</code> Fetches the caret that allows text-oriented navigation over the view.	36
Color	<code>getCaretColor()</code> Fetches the current color used to render the caret.	40
CaretListener[]	<code>getCaretListeners()</code> Returns an array of all the caret listeners registered on this text component.	34
int	<code>getCaretPosition()</code> Returns the position of the text insertion caret for the text component.	45
Color	<code>getDisabledTextColor()</code> Fetches the current color used to render the disabled text.	41
Document	<code>getDocument()</code> Fetches the model associated with the editor.	34
boolean	<code>getDragEnabled()</code> Returns whether or not automatic drag handling is enabled.	37
JTextComponent.DropLocation	<code>getDropLocation()</code> Returns the location that this component should visually indicate as the drop location during a DnD operation over the component, or null if no location is to currently be shown.	38
DropMode	<code>getDropMode()</code> Returns the drop mode for this component.	38
char	<code>getFocusAccelerator()</code> Returns the key accelerator that will cause the receiving text component to get the focus.	44
Highlighter	<code>getHighlighter()</code> Fetches the object responsible for making highlights.	36
InputMethodRequests	<code>getInputMethodRequests()</code>	53
Keymap	<code>getKeymap()</code> Fetches the keymap currently active in this text component.	38
static Keymap	<code>getKeymap(String nm)</code> Fetches a named keymap previously added to the document.	39

Insets	getMargin() Returns the margin between the text component's border and its text.	35
NavigationFilter	getNavigationFilter() Returns the NavigationFilter.	35
Dimension	getPreferredScrollableViewportSize() Returns the preferred size of the viewport for a view component.	48
Printable	getPrintable (MessageFormat headerFormat, MessageFormat footerFormat) Returns a Printable to use for printing the content of this JTextComponent.	52
int	getScrollableBlockIncrement (Rectangle visibleRect, int orientation, int direction) Components that display logical rows or columns should compute the scroll increment that will completely expose one block of rows or columns, depending on the value of orientation.	49
boolean	getScrollableTracksViewportHeight() Returns true if a viewport should always force the height of this Scrollable to match the height of the viewport.	50
boolean	getScrollableTracksViewportWidth() Returns true if a viewport should always force the width of this Scrollable to match the width of the viewport.	49
int	getScrollableUnitIncrement (Rectangle visibleRect, int orientation, int direction) Components that display logical rows or columns should compute the scroll increment that will completely expose one new row or column, depending on the value of orientation.	49
String	getSelectedText() Returns the selected text contained in this TextComponent.	46
Color	getSelectedTextColor() Fetches the current color used to render the selected text.	41
Color	getSelectionColor() Fetches the current color used to render the selection.	40
int	getSelectionEnd() Returns the selected text's end position.	47
int	getSelectionStart() Returns the selected text's start position.	47
String	getText() Returns the text contained in this TextComponent.	46
String	getText (int offs, int len) Fetches a portion of the text represented by the component.	42
String	getToolTipText (MouseEvent event) Returns the string to be used as the tooltip for event.	48
TextUI	getUI() Fetches the user-interface factory for this text-oriented editor.	33
boolean	isEditable() Returns the boolean indicating whether this TextComponent is editable or not.	46
static void	loadKeymap (Keymap map, JTextComponent.KeyBinding[] bindings, Action[] actions) Loads a keymap with a bunch of bindings.	39
Rectangle	modelToView (int pos) Converts the given location in the model to a place in the view coordinate system.	42
void	moveCaretPosition (int pos) Moves the caret to a new position, leaving behind a mark defined by the last time setCaretPosition was called.	43

protected String	 paramString () Returns a string representation of this JTextComponent.	53
void	 paste () Transfers the contents of the system clipboard into the associated text model.	43
boolean	 print () A convenience print method that displays a print dialog, and then prints this JTextComponent in <i>interactive</i> mode with no header or footer text.	50
boolean	 print (MessageFormat headerFormat, MessageFormat footerFormat) A convenience print method that displays a print dialog, and then prints this JTextComponent in <i>interactive</i> mode with the specified header and footer text.	50
boolean	 print (MessageFormat headerFormat, MessageFormat footerFormat, boolean showPrintDialog, PrintService service, PrintRequestAttributeSet attributes, boolean interactive) Prints the content of this JTextComponent.	51
protected void	 processInputMethodEvent (InputMethodEvent e)	53
void	 read (Reader in, Object desc) Initializes from a stream.	44
void	 removeCaretListener (CaretListener listener) Removes a caret listener.	33
static Keymap	 removeKeymap (String nm) Removes a named keymap previously added to the document.	39
void	 removeNotify ()	45
void	 replaceSelection (String content) Replaces the currently selected content with new content represented by the given string.	41
void	 select (int selectionStart, int selectionEnd) Selects the text between the specified start and end positions.	47
void	 selectAll () Selects all the text in the TextComponent.	48
void	 setCaret (Caret c) Sets the caret to be used.	36
void	 setCaretColor (Color c) Sets the current color used to render the caret.	40
void	 setCaretPosition (int position) Sets the position of the text insertion caret for the TextComponent.	45
void	 setComponentOrientation (ComponentOrientation o)	34
void	 setDisabledTextColor (Color c) Sets the current color used to render the disabled text.	41
void	 setDocument (Document doc) Associates the editor with a text document.	34
void	 setDragEnabled (boolean b) Turns on or off automatic drag handling.	37
void	 setDropMode (DropMode dropMode) Sets the drop mode for this component.	37
void	 setEditable (boolean b) Sets the specified boolean to indicate whether or not this TextComponent should be editable.	46
void	 setFocusAccelerator (char aKey) Sets the key accelerator that will cause the receiving text component to get the focus.	44
void	 setHighlighter (Highlighter h) Sets the highlighter to be used.	36

void	setKeymap (Keymap map) Sets the keymap to use for binding events to actions.	36
void	setMargin (Insets m) Sets margin space between the text component's border and its text.	35
void	setNavigationFilter (NavigationFilter filter) Sets the NavigationFilter.	35
void	setSelectedTextColor (Color c) Sets the current color used to render the selected text.	41
void	setSelectionColor (Color c) Sets the current color used to render the selection.	40
void	setSelectionEnd (int selectionEnd) Sets the selection end to the specified position.	47
void	setSelectionStart (int selectionStart) Sets the selection start to the specified position.	47
void	setText (String t) Sets the text of this TextComponent to the specified text.	45
void	setUI (TextUI ui) Sets the user-interface factory for this text-oriented editor.	33
void	updateUI () Reloads the pluggable UI.	33
int	viewToModel (Point pt) Converts the given place in the view coordinate system to the nearest representative location in the model.	42
void	write (Writer out) Stores the contents of the model into the given stream.	44

Field Detail

DEFAULT_KEYMAP

```
public static final String DEFAULT_KEYMAP
```

The default keymap that will be shared by all JTextComponent instances unless they have had a different keymap set.

FOCUS_ACCELERATOR_KEY

```
public static final String FOCUS_ACCELERATOR_KEY
```

The bound property name for the focus accelerator.

Constructor Detail

JTextComponent

```
public JTextComponent()
```

Creates a new JTextComponent. Listeners for caret events are established, and the pluggable UI installed. The component is marked as editable. No layout manager is used, because layout is managed by the view subsystem of text. The document model is set to null.

Method Detail

getUI

```
public TextUI getUI()
```

Fetches the user-interface factory for this text-oriented editor.

Returns:
the factory

setUI

```
public void setUI(TextUI ui)
```

Sets the user-interface factory for this text-oriented editor.

Parameters:
ui - the factory

updateUI

```
public void updateUI()
```

Reloads the pluggable UI. The key used to fetch the new interface is `getUIClassID()`. The type of the UI is `TextUI`. `invalidate` is called after setting the UI.

Overrides:
updateUI in class `JComponent`

addCaretListener

```
public void addCaretListener(CaretListener listener)
```

Adds a caret listener for notification of any changes to the caret.

Parameters:
listener - the listener to be added

See Also:
`CaretEvent`

removeCaretListener

```
public void removeCaretListener(CaretListener listener)
```

Removes a caret listener.

Parameters:
listener - the listener to be removed

See Also:
`CaretEvent`

getCaretListeners

```
public CaretListener[] getCaretListeners()
```

Returns an array of all the caret listeners registered on this text component.

Returns:

all of this component's `CaretListeners` or an empty array if no caret listeners are currently registered

Since:

1.4

See Also:

[addCaretListener\(CaretListener\)](#), [removeCaretListener\(CaretListener\)](#)

fireCaretUpdate

```
protected void fireCaretUpdate(CaretEvent e)
```

Notifies all listeners that have registered interest for notification on this event type. The event instance is lazily created using the parameters passed into the fire method. The listener list is processed in a last-to-first manner.

Parameters:

e - the event

See Also:

`EventListenerList`

setDocument

```
public void setDocument(Document doc)
```

Associates the editor with a text document. The currently registered factory is used to build a view for the document, which gets displayed by the editor after revalidation. A `PropertyChange` event ("document") is propagated to each listener.

Parameters:

doc - the document to display/edit

See Also:

[getDocument\(\)](#)

getDocument

```
public Document getDocument()
```

Fetches the model associated with the editor. This is primarily for the UI to get at the minimal amount of state required to be a text editor. Subclasses will return the actual type of the model which will typically be something that extends `Document`.

Returns:

the model

setComponentOrientation

```
public void setComponentOrientation(ComponentOrientation o)
```

Overrides:

setComponentOrientation in class Component

getActions

```
public Action[] getActions()
```

Fetches the command list for the editor. This is the list of commands supported by the plugged-in UI augmented by the collection of commands that the editor itself supports. These are useful for binding to events, such as in a keymap.

Returns:

the command list

setMargin

```
public void setMargin(Insets m)
```

Sets margin space between the text component's border and its text. The text component's default `Border` object will use this value to create the proper margin. However, if a non-default border is set on the text component, it is that `Border` object's responsibility to create the appropriate margin space (else this property will effectively be ignored). This causes a redraw of the component. A `PropertyChange` event ("margin") is sent to all listeners.

Parameters:

m - the space between the border and the text

getMargin

```
public Insets getMargin()
```

Returns the margin between the text component's border and its text.

Returns:

the margin

setNavigationFilter

```
public void setNavigationFilter(NavigationFilter filter)
```

Sets the `NavigationFilter`. `NavigationFilter` is used by `DefaultCaret` and the default cursor movement actions as a way to restrict the cursor movement.

Since:

1.4

getNavigationFilter

```
public NavigationFilter getNavigationFilter()
```

Returns the `NavigationFilter`. `NavigationFilter` is used by `DefaultCaret` and the default cursor movement actions as a way to restrict the cursor movement. A null return value implies the cursor movement and selection should not be restricted.

Returns:
the NavigationFilter

Since:
1.4

getCaret

```
public Caret getCaret()
```

Fetches the caret that allows text-oriented navigation over the view.

Returns:
the caret

setCaret

```
public void setCaret(Caret c)
```

Sets the caret to be used. By default this will be set by the UI that gets installed. This can be changed to a custom caret if desired. Setting the caret results in a PropertyChange event ("caret") being fired.

Parameters:
c - the caret

See Also:
[getCaret\(\)](#)

getHighlighter

```
public Highlighter getHighlighter()
```

Fetches the object responsible for making highlights.

Returns:
the highlighter

setHighlighter

```
public void setHighlighter(Highlighter h)
```

Sets the highlighter to be used. By default this will be set by the UI that gets installed. This can be changed to a custom highlighter if desired. The highlighter can be set to null to disable it. A PropertyChange event ("highlighter") is fired when a new highlighter is installed.

Parameters:
h - the highlighter

See Also:
[getHighlighter\(\)](#)

setKeymap

```
public void setKeymap(Keymap map)
```

Sets the keymap to use for binding events to actions. Setting to `null` effectively disables keyboard input. A `PropertyChangeEvent` ("keymap") is fired when a new keymap is installed.

Parameters:

map - the keymap

See Also:

[getKeymap\(\)](#)

setDragEnabled

```
public void setDragEnabled(boolean b)
```

Turns on or off automatic drag handling. In order to enable automatic drag handling, this property should be set to `true`, and the component's `TransferHandler` needs to be non-`null`. The default value of the `dragEnabled` property is `false`.

The job of honoring this property, and recognizing a user drag gesture, lies with the look and feel implementation, and in particular, the component's `TextUI`. When automatic drag handling is enabled, most look and feels (including those that subclass `BasicLookAndFeel`) begin a drag and drop operation whenever the user presses the mouse button over a selection and then moves the mouse a few pixels. Setting this property to `true` can therefore have a subtle effect on how selections behave.

If a look and feel is used that ignores this property, you can still begin a drag and drop operation by calling `exportAsDrag` on the component's `TransferHandler`.

Parameters:

b - whether or not to enable automatic drag handling

Throws:

`HeadlessException` - if b is true and `GraphicsEnvironment.isHeadless()` returns true

Since:

1.4

See Also:

`GraphicsEnvironment.isHeadless()`, [setDragEnabled\(\)](#),
`JComponent.setTransferHandler(TransferHandler)`, `TransferHandler`

getDragEnabled

```
public boolean getDragEnabled()
```

Returns whether or not automatic drag handling is enabled.

Returns:

the value of the `dragEnabled` property

Since:

1.4

See Also:

[setDragEnabled\(boolean\)](#)

setDropMode

```
public final void setDropMode(DropMode dropMode)
```

Sets the drop mode for this component. For backward compatibility, the default for this property is `DropMode.USE_SELECTION`. Usage of `DropMode.INSERT` is recommended, however, for an improved user experience. It offers similar behavior of dropping between text locations, but does so without affecting the actual text selection and caret location.

JTextComponents support the following drop modes:

- `DropMode.USE_SELECTION`
- `DropMode.INSERT`

The drop mode is only meaningful if this component has a `TransferHandler` that accepts drops.

Parameters:

`dropMode` - the drop mode to use

Throws:

`IllegalArgumentException` - if the drop mode is unsupported or null

Since:

1.6

See Also:

[getDropMode\(\)](#), [getDropLocation\(\)](#), `JComponent.setTransferHandler(TransferHandler)`, `TransferHandler`

getDropMode

```
public final DropMode getDropMode()
```

Returns the drop mode for this component.

Returns:

the drop mode for this component

Since:

1.6

See Also:

[setDropMode\(DropMode\)](#)

getDropLocation

```
public final JTextComponent.DropLocation getDropLocation()
```

Returns the location that this component should visually indicate as the drop location during a DnD operation over the component, or null if no location is to currently be shown.

This method is not meant for querying the drop location from a `TransferHandler`, as the drop location is only set after the `TransferHandler`'s `canImport` has returned and has allowed for the location to be shown.

When this property changes, a property change event with name "dropLocation" is fired by the component.

Returns:

the drop location

Since:

1.6

See Also:

[setDropMode\(DropMode\)](#), `TransferHandler.canImport(TransferHandler.TransferSupport)`

getKeymap

```
public Keymap getKeymap()
```

Fetches the keymap currently active in this text component.

Returns:
the keymap

addKeymap

```
public static Keymap addKeymap(String nm,  
                                Keymap parent)
```

Adds a new keymap into the keymap hierarchy. Keymap bindings resolve from bottom up so an attribute specified in a child will override an attribute specified in the parent.

Parameters:

`nm` - the name of the keymap (must be unique within the collection of named keymaps in the document); the name may be `null` if the keymap is unnamed, but the caller is responsible for managing the reference returned as an unnamed keymap can't be fetched by name
`parent` - the parent keymap; this may be `null` if unspecified bindings need not be resolved in some other keymap

Returns:
the keymap

removeKeymap

```
public static Keymap removeKeymap(String nm)
```

Removes a named keymap previously added to the document. Keymaps with `null` names may not be removed in this way.

Parameters:

`nm` - the name of the keymap to remove

Returns:
the keymap that was removed

getKeymap

```
public static Keymap getKeymap(String nm)
```

Fetches a named keymap previously added to the document. This does not work with `null`-named keymaps.

Parameters:

`nm` - the name of the keymap

Returns:
the keymap

loadKeymap

```
public static void loadKeymap(Keymap map,  
                               JTextComponent.KeyBinding[] bindings,  
                               Action[] actions)
```

Loads a keymap with a bunch of bindings. This can be used to take a static table of definitions and load them into some keymap. The following example illustrates an example of binding some keys to the cut, copy, and paste actions associated with a `JTextComponent`. A code fragment to accomplish this might look as follows:

```
static final JTextComponent.KeyBinding[] defaultBindings = {
```

```
new JTextComponent.KeyBinding(
    KeyStroke.getKeyStroke(KeyEvent.VK_C, InputEvent.CTRL_MASK),
    DefaultEditorKit.copyAction),
new JTextComponent.KeyBinding(
    KeyStroke.getKeyStroke(KeyEvent.VK_V, InputEvent.CTRL_MASK),
    DefaultEditorKit.pasteAction),
new JTextComponent.KeyBinding(
    KeyStroke.getKeyStroke(KeyEvent.VK_X, InputEvent.CTRL_MASK),
    DefaultEditorKit.cutAction),
};

JTextComponent c = new JTextPane();
Keymap k = c.getKeymap();
JTextComponent.loadKeymap(k, defaultBindings, c.getActions());
```

The sets of bindings and actions may be empty but must be non-null.

Parameters:

map - the keymap
bindings - the bindings
actions - the set of actions

getCaretColor

```
public Color getCaretColor()
```

Fetches the current color used to render the caret.

Returns:

the color

setCaretColor

```
public void setCaretColor(Color c)
```

Sets the current color used to render the caret. Setting to null effectively restores the default color. Setting the color results in a PropertyChange event ("caretColor") being fired.

Parameters:

c - the color

See Also:

[getCaretColor\(\)](#)

getSelectionColor

```
public Color getSelectionColor()
```

Fetches the current color used to render the selection.

Returns:

the color

setSelectionColor

```
public void setSelectionColor(Color c)
```

Sets the current color used to render the selection. Setting the color to `null` is the same as setting `Color.white`. Setting the color results in a `PropertyChange` event ("selectionColor").

Parameters:

`c` - the color

See Also:

[getSelectionColor\(\)](#)

getSelectedTextColor

```
public Color getSelectedTextColor()
```

Fetches the current color used to render the selected text.

Returns:

the color

setSelectedTextColor

```
public void setSelectedTextColor(Color c)
```

Sets the current color used to render the selected text. Setting the color to `null` is the same as `Color.black`. Setting the color results in a `PropertyChange` event ("selectedTextColor") being fired.

Parameters:

`c` - the color

See Also:

[getSelectedTextColor\(\)](#)

getDisabledTextColor

```
public Color getDisabledTextColor()
```

Fetches the current color used to render the disabled text.

Returns:

the color

setDisabledTextColor

```
public void setDisabledTextColor(Color c)
```

Sets the current color used to render the disabled text. Setting the color fires off a `PropertyChange` event ("disabledTextColor").

Parameters:

`c` - the color

See Also:

[getDisabledTextColor\(\)](#)

replaceSelection

```
public void replaceSelection(String content)
```

Replaces the currently selected content with new content represented by the given string. If there is no selection this amounts to an insert of the given text. If there is no replacement text this amounts to a removal of the current selection.

This is the method that is used by the default implementation of the action for inserting content that gets bound to the keymap actions.

This method is thread safe, although most Swing methods are not. Please see [How to Use Threads](#) for more information.

Parameters:

content - the content to replace the selection with

getText

```
public String getText(int offs,  
                      int len)  
    throws BadLocationException
```

Fetches a portion of the text represented by the component. Returns an empty string if length is 0.

Parameters:

offs - the offset ≥ 0

len - the length ≥ 0

Returns:

the text

Throws:

BadLocationException

BadLocationException - if the offset or length are invalid

modelToView

```
public Rectangle modelToView(int pos)  
    throws BadLocationException
```

Converts the given location in the model to a place in the view coordinate system. The component must have a positive size for this translation to be computed (i.e. layout cannot be computed until the component has been sized). The component does not have to be visible or painted.

Parameters:

pos - the position ≥ 0

Returns:

the coordinates as a rectangle, with (r.x, r.y) as the location in the coordinate system, or null if the component does not yet have a positive size.

Throws:

BadLocationException

BadLocationException - if the given position does not represent a valid location in the associated document

See Also:

TextUI.modelToView(JTextComponent, int)

viewToModel

```
public int viewToModel(Point pt)
```

Converts the given place in the view coordinate system to the nearest representative location in the model. The component must have a positive size for this translation to be computed (i.e. layout cannot be computed until the component has been sized). The component does not have to be visible or painted.

Parameters:

pt - the location in the view to translate

Returns:

the offset ≥ 0 from the start of the document, or -1 if the component does not yet have a positive size.

See Also:

`TextUI.viewToModel(JTextComponent, Point)`

cut

```
public void cut()
```

Transfers the currently selected range in the associated text model to the system clipboard, removing the contents from the model. The current selection is reset. Does nothing for null selections.

See Also:

`Toolkit.getSystemClipboard(), Clipboard`

copy

```
public void copy()
```

Transfers the currently selected range in the associated text model to the system clipboard, leaving the contents in the text model. The current selection remains intact. Does nothing for null selections.

See Also:

`Toolkit.getSystemClipboard(), Clipboard`

paste

```
public void paste()
```

Transfers the contents of the system clipboard into the associated text model. If there is a selection in the associated view, it is replaced with the contents of the clipboard. If there is no selection, the clipboard contents are inserted in front of the current insert position in the associated view. If the clipboard is empty, does nothing.

See Also:

`replaceSelection(String), Toolkit.getSystemClipboard(), Clipboard`

moveCaretPosition

```
public void moveCaretPosition(int pos)
```

Moves the caret to a new position, leaving behind a mark defined by the last time `setCaretPosition` was called. This forms a selection. If the document is null, does nothing. The position must be between 0 and the length of the component's text or else an exception is thrown.

Parameters:

pos - the position

Throws:

`IllegalArgumentException` - if the value supplied for `position` is less than zero or greater than the component's text length

See Also:

[setCaretPosition\(int\)](#)

setFocusAccelerator

```
public void setFocusAccelerator(char aKey)
```

Sets the key accelerator that will cause the receiving text component to get the focus. The accelerator will be the key combination of the *alt* key and the character given (converted to upper case). By default, there is no focus accelerator key. Any previous key accelerator setting will be superseded. A `\0` key setting will be registered, and has the effect of turning off the focus accelerator. When the new key is set, a `PropertyChange` event (`FOCUS_ACCELERATOR_KEY`) will be fired.

Parameters:

`aKey` - the key

See Also:

[getFocusAccelerator\(\)](#)

getFocusAccelerator

```
public char getFocusAccelerator()
```

Returns the key accelerator that will cause the receiving text component to get the focus. Return `\0` if no focus accelerator has been set.

Returns:

the key

read

```
public void read(Reader in,  
                Object desc)  
    throws IOException
```

Initializes from a stream. This creates a model of the type appropriate for the component and initializes the model from the stream. By default this will load the model as plain text. Previous contents of the model are discarded.

Parameters:

`in` - the stream to read from

`desc` - an object describing the stream; this might be a string, a `File`, a `URL`, etc. Some kinds of documents (such as html for example) might be able to make use of this information; if non-null, it is added as a property of the document

Throws:

`IOException` - as thrown by the stream being used to initialize

See Also:

`EditorKit.createDefaultDocument()`, [setDocument\(Document\)](#), `PlainDocument`

write

```
public void write(Writer out)  
    throws IOException
```

Stores the contents of the model into the given stream. By default this will store the model as plain text.

Parameters:

out - the output stream

Throws:

IOException - on any I/O error

removeNotify

```
public void removeNotify()
```

Overrides:

removeNotify in class JComponent

setCaretPosition

```
public void setCaretPosition(int position)
```

Sets the position of the text insertion caret for the `TextComponent`. Note that the caret tracks change, so this may move if the underlying text of the component is changed. If the document is `null`, does nothing. The position must be between 0 and the length of the component's text or else an exception is thrown.

Parameters:

position - the position

Throws:

IllegalArgumentException - if the value supplied for position is less than zero or greater than the component's text length

getCaretPosition

```
public int getCaretPosition()
```

Returns the position of the text insertion caret for the text component.

Returns:

the position of the text insertion caret for the text component ≥ 0

setText

```
public void setText(String t)
```

Sets the text of this `TextComponent` to the specified text. If the text is `null` or empty, has the effect of simply deleting the old text. When text has been inserted, the resulting caret location is determined by the implementation of the caret class.

This method is thread safe, although most Swing methods are not. Please see [How to Use Threads](#) for more information. Note that text is not a bound property, so no `PropertyChangeEvent` is fired when it changes. To listen for changes to the text, use `DocumentListener`.

Parameters:

t - the new text to be set

See Also:

[getText\(int, int\)](#), `DefaultCaret`

getText

```
public String getText()
```

Returns the text contained in this `TextComponent`. If the underlying document is null, will give a `NullPointerException`. Note that text is not a bound property, so no `PropertyChangeEvent` is fired when it changes. To listen for changes to the text, use `DocumentListener`.

Returns:

the text

Throws:

`NullPointerException` - if the document is null

See Also:

[setText\(String\)](#)

getSelectedText

```
public String getSelectedText()
```

Returns the selected text contained in this `TextComponent`. If the selection is null or the document empty, returns null.

Returns:

the text

Throws:

`IllegalArgumentException` - if the selection doesn't have a valid mapping into the document for some reason

See Also:

[setText\(String\)](#)

isEditable

```
public boolean isEditable()
```

Returns the boolean indicating whether this `TextComponent` is editable or not.

Returns:

the boolean value

See Also:

[setEditable\(boolean\)](#)

setEditable

```
public void setEditable(boolean b)
```

Sets the specified boolean to indicate whether or not this `TextComponent` should be editable. A `PropertyChange` event ("editable") is fired when the state is changed.

Parameters:

b - the boolean to be set

See Also:

[isEditable\(\)](#)

getSelectionStart

```
public int getSelectionStart()
```

Returns the selected text's start position. Return 0 for an empty document, or the value of dot if no selection.

Returns:

the start position ≥ 0

setSelectionStart

```
public void setSelectionStart(int selectionStart)
```

Sets the selection start to the specified position. The new starting point is constrained to be before or at the current selection end.

This is available for backward compatibility to code that called this method on `java.awt.TextComponent`. This is implemented to forward to the `Caret` implementation which is where the actual selection is maintained.

Parameters:

selectionStart - the start position of the text ≥ 0

getSelectionEnd

```
public int getSelectionEnd()
```

Returns the selected text's end position. Return 0 if the document is empty, or the value of dot if there is no selection.

Returns:

the end position ≥ 0

setSelectionEnd

```
public void setSelectionEnd(int selectionEnd)
```

Sets the selection end to the specified position. The new end point is constrained to be at or after the current selection start.

This is available for backward compatibility to code that called this method on `java.awt.TextComponent`. This is implemented to forward to the `Caret` implementation which is where the actual selection is maintained.

Parameters:

selectionEnd - the end position of the text ≥ 0

select

```
public void select(int selectionStart,  
                  int selectionEnd)
```

Selects the text between the specified start and end positions.

This method sets the start and end positions of the selected text, enforcing the restriction that the start position must be greater than or equal to zero. The end position must be greater than or equal to the start position, and less than or equal to the length of the text component's text.

If the caller supplies values that are inconsistent or out of bounds, the method enforces these constraints silently, and without failure. Specifically, if the start position or end position is greater than the length of the text, it is reset to equal the text length. If the start position is less than zero, it is reset to zero, and if the end position is less than the start position, it is reset to the start position.

This call is provided for backward compatibility. It is routed to a call to `setCaretPosition` followed by a call to `moveCaretPosition`. The preferred way to manage selection is by calling those methods directly.

Parameters:

`selectionStart` - the start position of the text
`selectionEnd` - the end position of the text

See Also:

[setCaretPosition\(int\)](#), [moveCaretPosition\(int\)](#)

selectAll

```
public void selectAll()
```

Selects all the text in the `TextComponent`. Does nothing on a `null` or empty document.

getToolTipText

```
public String getToolTipText(MouseEvent event)
```

Returns the string to be used as the tooltip for event. This will return one of:

1. If `setToolTipText` has been invoked with a non-null value, it will be returned, otherwise
2. The value from invoking `getToolTipText` on the UI will be returned.

By default `JTextComponent` does not register itself with the `ToolTipManager`. This means that tooltips will NOT be shown from the `TextUI` unless `registerComponent` has been invoked on the `ToolTipManager`.

Overrides:

`getToolTipText` in class `JComponent`

Parameters:

`event` - the event in question

Returns:

the string to be used as the tooltip for event

See Also:

`JComponent.setToolTipText(String)`, `TextUI.getToolTipText(JTextComponent, Point)`,
`ToolTipManager.registerComponent(JComponent)`

getPreferredScrollableViewportSize

```
public Dimension getPreferredScrollableViewportSize()
```

Returns the preferred size of the viewport for a view component. This is implemented to do the default behavior of returning the preferred size of the component.

Specified by:

`getPreferredScrollableViewportSize` in interface `Scrollable`

Returns:

the preferredSize of a `JViewport` whose view is this `Scrollable`

getScrollableUnitIncrement

```
public int getScrollableUnitIncrement(Rectangle visibleRect,  
                                     int orientation,  
                                     int direction)
```

Components that display logical rows or columns should compute the scroll increment that will completely expose one new row or column, depending on the value of orientation. Ideally, components should handle a partially exposed row or column by returning the distance required to completely expose the item.

The default implementation of this is to simply return 10% of the visible area. Subclasses are likely to be able to provide a much more reasonable value.

Specified by:

getScrollableUnitIncrement in interface Scrollable

Parameters:

visibleRect - the view area visible within the viewport
orientation - either SwingConstants.VERTICAL or SwingConstants.HORIZONTAL
direction - less than zero to scroll up/left, greater than zero for down/right

Returns:

the "unit" increment for scrolling in the specified direction

Throws:

IllegalArgumentException - for an invalid orientation

See Also:

JScrollBar.setUnitIncrement(int)

getScrollableBlockIncrement

```
public int getScrollableBlockIncrement(Rectangle visibleRect,  
                                       int orientation,  
                                       int direction)
```

Components that display logical rows or columns should compute the scroll increment that will completely expose one block of rows or columns, depending on the value of orientation.

The default implementation of this is to simply return the visible area. Subclasses will likely be able to provide a much more reasonable value.

Specified by:

getScrollableBlockIncrement in interface Scrollable

Parameters:

visibleRect - the view area visible within the viewport
orientation - either SwingConstants.VERTICAL or SwingConstants.HORIZONTAL
direction - less than zero to scroll up/left, greater than zero for down/right

Returns:

the "block" increment for scrolling in the specified direction

Throws:

IllegalArgumentException - for an invalid orientation

See Also:

JScrollBar.setBlockIncrement(int)

getScrollableTracksViewportWidth

```
public boolean getScrollableTracksViewportWidth()
```

Returns true if a viewport should always force the width of this Scrollable to match the width of the viewport. For example a normal text view that supported line wrapping would return true here, since it would be undesirable for

wrapped lines to disappear beyond the right edge of the viewport. Note that returning true for a `Scrollable` whose ancestor is a `JScrollPane` effectively disables horizontal scrolling.

Scrolling containers, like `JViewport`, will use this method each time they are validated.

Specified by:

`getScrollableTracksViewportWidth` in interface `Scrollable`

Returns:

true if a viewport should force the `Scrollables` width to match its own

getScrollableTracksViewportHeight

```
public boolean getScrollableTracksViewportHeight()
```

Returns true if a viewport should always force the height of this `Scrollable` to match the height of the viewport. For example a columnar text view that flowed text in left to right columns could effectively disable vertical scrolling by returning true here.

Scrolling containers, like `JViewport`, will use this method each time they are validated.

Specified by:

`getScrollableTracksViewportHeight` in interface `Scrollable`

Returns:

true if a viewport should force the `Scrollables` height to match its own

print

```
public boolean print()
    throws PrinterException
```

A convenience print method that displays a print dialog, and then prints this `JTextComponent` in *interactive* mode with no header or footer text. Note: this method blocks until printing is done.

Note: In *headless* mode, no dialogs will be shown.

This method calls the full featured `print` method to perform printing.

Returns:

true, unless printing is canceled by the user

Throws:

`PrinterException` - if an error in the print system causes the job to be aborted
`SecurityException` - if this thread is not allowed to initiate a print job request

Since:

1.6

See Also:

`print(MessageFormat, MessageFormat, boolean, PrintService, PrintRequestAttributeSet, boolean)`

print

```
public boolean print(MessageFormat headerFormat,
    MessageFormat footerFormat)
    throws PrinterException
```

A convenience print method that displays a print dialog, and then prints this `JTextComponent` in *interactive* mode with the specified header and footer text. Note: this method blocks until printing is done.

Note: In *headless* mode, no dialogs will be shown.

This method calls the full featured [print](#) method to perform printing.

Parameters:

`headerFormat` - the text, in `MessageFormat`, to be used as the header, or `null` for no header

`footerFormat` - the text, in `MessageFormat`, to be used as the footer, or `null` for no footer

Returns:

`true`, unless printing is canceled by the user

Throws:

`PrinterException` - if an error in the print system causes the job to be aborted

`SecurityException` - if this thread is not allowed to initiate a print job request

Since:

1.6

See Also:

[print\(MessageFormat, MessageFormat, boolean, PrintService, PrintRequestAttributeSet, boolean\)](#), [MessageFormat](#)

print

```
public boolean print(MessageFormat headerFormat,
                    MessageFormat footerFormat,
                    boolean showPrintDialog,
                    PrintService service,
                    PrintRequestAttributeSet attributes,
                    boolean interactive)
    throws PrinterException
```

Prints the content of this `JTextComponent`. Note: this method blocks until printing is done.

Page header and footer text can be added to the output by providing `MessageFormat` arguments. The printing code requests `Strings` from the formats, providing a single item which may be included in the formatted string: an `Integer` representing the current page number.

`showPrintDialog` `boolean` parameter allows you to specify whether a print dialog is displayed to the user. When it is, the user may use the dialog to change printing attributes or even cancel the print.

`service` allows you to provide the initial `PrintService` for the print dialog, or to specify `PrintService` to print to when the dialog is not shown.

`attributes` can be used to provide the initial values for the print dialog, or to supply any needed attributes when the dialog is not shown. `attributes` can be used to control how the job will print, for example *duplex* or *single-sided*.

`interactive` `boolean` parameter allows you to specify whether to perform printing in *interactive* mode. If `true`, a progress dialog, with an abort option, is displayed for the duration of printing. This dialog is *modal* when `print` is invoked on the *Event Dispatch Thread* and *non-modal* otherwise. **Warning:** calling this method on the *Event Dispatch Thread* with `interactive` `false` blocks *all* events, including repaints, from being processed until printing is complete. It is only recommended when printing from an application with no visible GUI.

Note: In *headless* mode, `showPrintDialog` and `interactive` parameters are ignored and no dialogs are shown.

This method ensures the document is not mutated during printing. To indicate it visually, `setEnabled(false)` is set for the duration of printing.

This method uses [getPrintable\(MessageFormat, MessageFormat\)](#) to render document content.

This method is thread-safe, although most Swing methods are not. Please see [How to Use Threads](#) for more information.

Sample Usage. This code snippet shows a cross-platform print dialog and then prints the `JTextComponent` in *interactive* mode unless the user cancels the dialog:

```
textComponent.print(new MessageFormat("My text component header"),
    new MessageFormat("Footer. Page - {0}"), true, null, null, true);
```

Executing this code off the *Event Dispatch Thread* performs printing on the *background*. The following pattern might be used for *background* printing:

```
FutureTask<Boolean> future =
    new FutureTask<Boolean>(
        new Callable<Boolean>() {
            public Boolean call() {
                return textComponent.print(...);
            }
        });
executor.execute(future);
```

Parameters:

`headerFormat` - the text, in `MessageFormat`, to be used as the header, or null for no header
`footerFormat` - the text, in `MessageFormat`, to be used as the footer, or null for no footer
`showPrintDialog` - true to display a print dialog, false otherwise
`service` - initial `PrintService`, or null for the default
`attributes` - the job attributes to be applied to the print job, or null for none
`interactive` - whether to print in an interactive mode

Returns:

true, unless printing is canceled by the user

Throws:

`PrinterException` - if an error in the print system causes the job to be aborted
`SecurityException` - if this thread is not allowed to initiate a print job request

Since:

1.6

See Also:

[getPrintable\(MessageFormat, MessageFormat\)](#), `MessageFormat`,
`GraphicsEnvironment.isHeadless()`, `FutureTask`

getPrintable

```
public Printable getPrintable(MessageFormat headerFormat,
    MessageFormat footerFormat)
```

Returns a `Printable` to use for printing the content of this `JTextComponent`. The returned `Printable` prints the document as it looks on the screen except being reformatted to fit the paper. The returned `Printable` can be wrapped inside another `Printable` in order to create complex reports and documents.

The returned `Printable` shares the document with this `JTextComponent`. It is the responsibility of the developer to ensure that the document is not mutated while this `Printable` is used. Printing behavior is undefined when the document is mutated during printing.

Page header and footer text can be added to the output by providing `MessageFormat` arguments. The printing code requests `Strings` from the formats, providing a single item which may be included in the formatted string: an `Integer` representing the current page number.

The returned `Printable` when printed, formats the document content appropriately for the page size. For correct line wrapping the imageable width of all pages must be the same. See `PageFormat.getImageableWidth()`.

This method is thread-safe, although most Swing methods are not. Please see [How to Use Threads](#) for more information.

The returned `Printable` can be printed on any thread.

This implementation returned Printable performs all painting on the *Event Dispatch Thread*, regardless of what thread it is used on.

Parameters:

headerFormat - the text, in MessageFormat, to be used as the header, or null for no header
footerFormat - the text, in MessageFormat, to be used as the footer, or null for no footer

Returns:

a Printable for use in printing content of this JTextComponent

Since:

1.6

See Also:

Printable, PageFormat, [Document.render\(java.lang.Runnable\)](#)

getAccessibleContext

```
public AccessibleContext getAccessibleContext()
```

Gets the AccessibleContext associated with this JTextComponent. For text components, the AccessibleContext takes the form of an AccessibleJTextComponent. A new AccessibleJTextComponent instance is created if necessary.

Specified by:

getAccessibleContext in interface Accessible

Overrides:

getAccessibleContext in class JComponent

Returns:

an AccessibleJTextComponent that serves as the AccessibleContext of this JTextComponent

paramString

```
protected String paramString()
```

Returns a string representation of this JTextComponent. This method is intended to be used only for debugging purposes, and the content and format of the returned string may vary between implementations. The returned string may be empty but may not be null.

Overriding paramString to provide information about the specific new aspects of the JFC components.

Overrides:

paramString in class JComponent

Returns:

a string representation of this JTextComponent

processInputMethodEvent

```
protected void processInputMethodEvent(InputMethodEvent e)
```

Overrides:

processInputMethodEvent in class Component

getInputMethodRequests

```
public InputMethodRequests getInputMethodRequests()
```

Overrides:

getInputMethodRequests in class Component

addInputMethodListener

public void **addInputMethodListener**(InputMethodListener l)

Overrides:

addInputMethodListener in class Component

javax.swing.text

JTextComponent.AccessibleJTextComponent

```

java.lang.Object
├── javax.accessibility.AccessibleContext
│   ├── java.awt.Component.AccessibleAWTComponent
│   │   ├── java.awt.Container.AccessibleAWTContainer
│   │   │   ├── javax.swing.JComponent.AccessibleJComponent
│   │   │   │   └── javax.swing.text.JTextComponent.AccessibleJTextComponent

```

All Implemented Interfaces:

AccessibleText, CaretListener, EventListener, DocumentListener, AccessibleAction, AccessibleEditableText, AccessibleExtendedText, AccessibleExtendedComponent, AccessibleComponent, Serializable

Enclosing class:

[JTextComponent](#)

```

public class JTextComponent.AccessibleJTextComponent
extends JComponent.AccessibleJComponent
implements AccessibleText, CaretListener, DocumentListener, AccessibleAction,
AccessibleEditableText, AccessibleExtendedText

```

This class implements accessibility support for the JTextComponent class. It provides an implementation of the Java Accessibility API appropriate to menu user-interface elements.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the java.beans package. Please see XMLEncoder.

Constructor Summary		Page
JTextComponent.AccessibleJTextComponent()	Constructs an AccessibleJTextComponent.	57

Method Summary		Page
void	caretUpdate (CaretEvent e) Handles caret updates (fire appropriate property change event, which are AccessibleContext.ACCESSIBLE_CARET_PROPERTY and AccessibleContext.ACCESSIBLE_SELECTION_PROPERTY).	57
void	changedUpdate (DocumentEvent e) Handles document remove (fire appropriate property change event, which is AccessibleContext.ACCESSIBLE_TEXT_PROPERTY).	58
void	cut (int startIndex, int endIndex) Cuts the text between two indices into the system clipboard.	63
void	delete (int startIndex, int endIndex) Deletes the text between two indices	63
boolean	doAccessibleAction (int i) Performs the specified Action on the object	66
AccessibleAction	getAccessibleAction ()	66
int	getAccessibleActionCount () Returns the number of accessible actions available in this object If there are more than one, the first one is considered the "default" action of the object.	66

String	getAccessibleActionDescription (int i) Returns a description of the specified action of the object.	66
AccessibleEditableText	getAccessibleEditableText () Returns the AccessibleEditableText interface for this text component.	61
AccessibleRole	getAccessibleRole () Gets the role of this object.	58
AccessibleStateSet	getAccessibleStateSet () Gets the state set of the JTextComponent.	58
AccessibleText	getAccessibleText () Get the AccessibleText associated with this object.	58
String	getAfterIndex (int part, int index) Returns the String after a given index.	61
String	getAtIndex (int part, int index) Returns the String at a given index.	61
String	getBeforeIndex (int part, int index) Returns the String before a given index.	61
int	getCaretPosition () Returns the zero-based offset of the caret.	59
AttributeSet	getCharacterAttribute (int i) Returns the AttributeSet for a given character (at a given index).	60
Rectangle	getCharacterBounds (int i) Determines the bounding box of the character at the given index into the string.	59
int	getCharCount () Returns the number of characters (valid indices)	59
int	getIndexAtPoint (Point p) Given a point in local coordinates, return the zero-based index of the character under that Point.	59
String	getSelectedText () Returns the portion of the text that is selected.	60
int	getSelectionEnd () Returns the end offset within the selected text.	60
int	getSelectionStart () Returns the start offset within the selected text.	60
Rectangle	getTextBounds (int startIndex, int endIndex) Returns the Rectangle enclosing the text between two indices.	65
String	getTextRange (int startIndex, int endIndex) Returns the text string between two indices.	62
AccessibleTextSequence	getTextSequenceAfter (int part, int index) Returns the AccessibleTextSequence after a given index.	65
AccessibleTextSequence	getTextSequenceAt (int part, int index) Returns the AccessibleTextSequence at a given index.	64
AccessibleTextSequence	getTextSequenceBefore (int part, int index) Returns the AccessibleTextSequence before a given index.	65
void	insertTextAtIndex (int index, String s) Inserts the specified string at the given index	62
void	insertUpdate (DocumentEvent e) Handles document insert (fire appropriate property change event which is AccessibleContext.ACCESSIBLE_TEXT_PROPERTY).	57
void	paste (int startIndex) Pastes the text from the system clipboard into the text starting at the specified index.	63
void	removeUpdate (DocumentEvent e) Handles document remove (fire appropriate property change event, which is AccessibleContext.ACCESSIBLE_TEXT_PROPERTY).	57

void	replaceText (int startIndex, int endIndex, String s) Replaces the text between two indices with the specified string.	63
void	selectText (int startIndex, int endIndex) Selects the text between two indices.	64
void	setAttributes (int startIndex, int endIndex, AttributeSet as) Sets attributes for the text between two indices.	64
void	setTextContents (String s) Sets the text contents to the specified string.	62

Constructor Detail

JTextComponent.AccessibleJTextComponent

```
public JTextComponent.AccessibleJTextComponent()
```

Constructs an AccessibleJTextComponent. Adds a listener to track caret change.

Method Detail

caretUpdate

```
public void caretUpdate(CaretEvent e)
```

Handles caret updates (fire appropriate property change event, which are AccessibleContext.ACCESSIBLE_CARET_PROPERTY and AccessibleContext.ACCESSIBLE_SELECTION_PROPERTY). This keeps track of the dot position internally. When the caret moves, the internal position is updated after firing the event.

Specified by:

caretUpdate in interface CaretListener

Parameters:

e - the CaretEvent

insertUpdate

```
public void insertUpdate(DocumentEvent e)
```

Handles document insert (fire appropriate property change event which is AccessibleContext.ACCESSIBLE_TEXT_PROPERTY). This tracks the changed offset via the event.

Specified by:

insertUpdate in interface DocumentListener

Parameters:

e - the DocumentEvent

removeUpdate

```
public void removeUpdate(DocumentEvent e)
```

Handles document remove (fire appropriate property change event, which is AccessibleContext.ACCESSIBLE_TEXT_PROPERTY). This tracks the changed offset via the event.

Specified by:

removeUpdate in interface DocumentListener

Parameters:

e - the DocumentEvent

changedUpdate

```
public void changedUpdate(DocumentEvent e)
```

Handles document remove (fire appropriate property change event, which is `AccessibleContext.ACCESSIBLE_TEXT_PROPERTY`). This tracks the changed offset via the event.

Specified by:

changedUpdate in interface DocumentListener

Parameters:

e - the DocumentEvent

getAccessibleStateSet

```
public AccessibleStateSet getAccessibleStateSet()
```

Gets the state set of the JTextComponent. The AccessibleStateSet of an object is composed of a set of unique AccessibleState's. A change in the AccessibleStateSet of an object will cause a PropertyChangeEvent to be fired for the `AccessibleContext.ACCESSIBLE_STATE_PROPERTY` property.

Overrides:

getAccessibleStateSet in class JComponent.AccessibleJComponent

Returns:

an instance of AccessibleStateSet containing the current state set of the object

See Also:

AccessibleStateSet, AccessibleState,
JComponent.AccessibleJComponent.addPropertyChangeListener(PropertyChangeListener)

getAccessibleRole

```
public AccessibleRole getAccessibleRole()
```

Gets the role of this object.

Overrides:

getAccessibleRole in class JComponent.AccessibleJComponent

Returns:

an instance of AccessibleRole describing the role of the object (`AccessibleRole.TEXT`)

See Also:

AccessibleRole

getAccessibleText

```
public AccessibleText getAccessibleText()
```

Get the AccessibleText associated with this object. In the implementation of the Java Accessibility API for this class, return this object, which is responsible for implementing the AccessibleText interface on behalf of itself.

Overrides:

getAccessibleText in class AccessibleContext

Returns:

this object

getIndexAtPoint

```
public int getIndexAtPoint(Point p)
```

Given a point in local coordinates, return the zero-based index of the character under that Point. If the point is invalid, this method returns -1.

Specified by:

getIndexAtPoint in interface AccessibleText

Parameters:

p - the Point in local coordinates

Returns:

the zero-based index of the character under Point p.

getCharacterBounds

```
public Rectangle getCharacterBounds(int i)
```

Determines the bounding box of the character at the given index into the string. The bounds are returned in local coordinates. If the index is invalid a null rectangle is returned. The screen coordinates returned are "unscrolled coordinates" if the JTextComponent is contained in a JScrollPane in which case the resulting rectangle should be composed with the parent coordinates. A good algorithm to use is: Accessible a: AccessibleText at = a.getAccessibleText(); AccessibleComponent ac = a.getAccessibleComponent(); Rectangle r = at.getCharacterBounds(); Point p = ac.getLocation(); r.x += p.x; r.y += p.y; Note: the JTextComponent must have a valid size (e.g. have been added to a parent container whose ancestor container is a valid top-level window) for this method to be able to return a meaningful (non-null) value.

Specified by:

getCharacterBounds in interface AccessibleText

Parameters:

i - the index into the String >= 0

Returns:

the screen coordinates of the character's bounding box

getCharCount

```
public int getCharCount()
```

Returns the number of characters (valid indices)

Specified by:

getCharCount in interface AccessibleText

Returns:

the number of characters >= 0

getCaretPosition

```
public int getCaretPosition()
```

Returns the zero-based offset of the caret. Note: The character to the right of the caret will have the same index value as the offset (the caret is between two characters).

Specified by:

`getCaretPosition` in interface `AccessibleText`

Returns:

the zero-based offset of the caret.

getCharacterAttribute

```
public AttributeSet getCharacterAttribute(int i)
```

Returns the AttributeSet for a given character (at a given index).

Specified by:

`getCharacterAttribute` in interface `AccessibleText`

Parameters:

`i` - the zero-based index into the text

Returns:

the AttributeSet of the character

getSelectionStart

```
public int getSelectionStart()
```

Returns the start offset within the selected text. If there is no selection, but there is a caret, the start and end offsets will be the same. Return 0 if the text is empty, or the caret position if no selection.

Specified by:

`getSelectionStart` in interface `AccessibleText`

Returns:

the index into the text of the start of the selection ≥ 0

getSelectionEnd

```
public int getSelectionEnd()
```

Returns the end offset within the selected text. If there is no selection, but there is a caret, the start and end offsets will be the same. Return 0 if the text is empty, or the caret position if no selection.

Specified by:

`getSelectionEnd` in interface `AccessibleText`

Returns:

the index into the text of the end of the selection ≥ 0

getSelectedText

```
public String getSelectedText()
```

Returns the portion of the text that is selected.

Specified by:

`getSelectedText` in interface `AccessibleText`

Returns:
the text, null if no selection

getAtIndex

```
public String getAtIndex(int part,  
                          int index)
```

Returns the String at a given index. Whitespace between words is treated as a word.

Specified by:
getAtIndex in interface AccessibleText

Parameters:
part - the CHARACTER, WORD, or SENTENCE to retrieve
index - an index within the text

Returns:
the letter, word, or sentence.

getAfterIndex

```
public String getAfterIndex(int part,  
                              int index)
```

Returns the String after a given index. Whitespace between words is treated as a word.

Specified by:
getAfterIndex in interface AccessibleText

Parameters:
part - the CHARACTER, WORD, or SENTENCE to retrieve
index - an index within the text

Returns:
the letter, word, or sentence.

getBeforeIndex

```
public String getBeforeIndex(int part,  
                               int index)
```

Returns the String before a given index. Whitespace between words is treated a word.

Specified by:
getBeforeIndex in interface AccessibleText

Parameters:
part - the CHARACTER, WORD, or SENTENCE to retrieve
index - an index within the text

Returns:
the letter, word, or sentence.

getAccessibleEditableText

```
public AccessibleEditableText getAccessibleEditableText()
```

Returns the AccessibleEditableText interface for this text component.

Overrides:

getAccessibleEditableText in class AccessibleContext

Returns:

the AccessibleEditableText interface

Since:

1.4

setTextContents

```
public void setTextContents(String s)
```

Sets the text contents to the specified string.

Specified by:

setTextContents in interface AccessibleEditableText

Parameters:

s - the string to set the text contents

Since:

1.4

insertTextAtIndex

```
public void insertTextAtIndex(int index,  
                               String s)
```

Inserts the specified string at the given index

Specified by:

insertTextAtIndex in interface AccessibleEditableText

Parameters:

index - the index in the text where the string will be inserted

s - the string to insert in the text

Since:

1.4

getTextRange

```
public String getTextRange(int startIndex,  
                             int endIndex)
```

Returns the text string between two indices.

Specified by:

getTextRange in interface AccessibleEditableText

getTextRange in interface AccessibleExtendedText

Parameters:

startIndex - the starting index in the text

endIndex - the ending index in the text

Returns:

the text string between the indices

Since:

1.4

delete

```
public void delete(int startIndex,  
                  int endIndex)
```

Deletes the text between two indices

Specified by:

delete in interface AccessibleEditableText

Parameters:

startIndex - the starting index in the text

endIndex - the ending index in the text

Since:

1.4

cut

```
public void cut(int startIndex,  
               int endIndex)
```

Cuts the text between two indices into the system clipboard.

Specified by:

cut in interface AccessibleEditableText

Parameters:

startIndex - the starting index in the text

endIndex - the ending index in the text

Since:

1.4

paste

```
public void paste(int startIndex)
```

Pastes the text from the system clipboard into the text starting at the specified index.

Specified by:

paste in interface AccessibleEditableText

Parameters:

startIndex - the starting index in the text

Since:

1.4

replaceText

```
public void replaceText(int startIndex,  
                        int endIndex,  
                        String s)
```

Replaces the text between two indices with the specified string.

Specified by:

replaceText in interface AccessibleEditableText

Parameters:

startIndex - the starting index in the text

endIndex - the ending index in the text

s - the string to replace the text between two indices

Since:

1.4

selectText

```
public void selectText(int startIndex,  
                       int endIndex)
```

Selects the text between two indices.

Specified by:

selectText in interface AccessibleEditableText

Parameters:

startIndex - the starting index in the text

endIndex - the ending index in the text

Since:

1.4

setAttributes

```
public void setAttributes(int startIndex,  
                          int endIndex,  
                          AttributeSet as)
```

Sets attributes for the text between two indices.

Specified by:

setAttributes in interface AccessibleEditableText

Parameters:

startIndex - the starting index in the text

endIndex - the ending index in the text

as - the attribute set

Since:

1.4

See Also:

AttributeSet

getTextSequenceAt

```
public AccessibleTextSequence getTextSequenceAt(int part,  
                                                  int index)
```

Returns the AccessibleTextSequence at a given index.

Specified by:

getTextSequenceAt in interface AccessibleExtendedText

Parameters:

part - the CHARACTER, WORD, SENTENCE, LINE or ATTRIBUTE_RUN to retrieve

index - an index within the text

Returns:

an AccessibleTextSequence specifying the text if part and index are valid. Otherwise, null is returned

Since:

1.6

See Also:

`AccessibleText.CHARACTER`, `AccessibleText.WORD`, `AccessibleText.SENTENCE`,
`AccessibleExtendedText.LINE`, `AccessibleExtendedText.ATTRIBUTE_RUN`

getTextSequenceAfter

```
public AccessibleTextSequence getTextSequenceAfter(int part,  
                                                    int index)
```

Returns the `AccessibleTextSequence` after a given index.

Specified by:

`getTextSequenceAfter` in interface `AccessibleExtendedText`

Parameters:

`part` - the `CHARACTER`, `WORD`, `SENTENCE`, `LINE` or `ATTRIBUTE_RUN` to retrieve
`index` - an index within the text

Returns:

an `AccessibleTextSequence` specifying the text if `part` and `index` are valid. Otherwise, `null` is returned

Since:

1.6

See Also:

`AccessibleText.CHARACTER`, `AccessibleText.WORD`, `AccessibleText.SENTENCE`,
`AccessibleExtendedText.LINE`, `AccessibleExtendedText.ATTRIBUTE_RUN`

getTextSequenceBefore

```
public AccessibleTextSequence getTextSequenceBefore(int part,  
                                                    int index)
```

Returns the `AccessibleTextSequence` before a given index.

Specified by:

`getTextSequenceBefore` in interface `AccessibleExtendedText`

Parameters:

`part` - the `CHARACTER`, `WORD`, `SENTENCE`, `LINE` or `ATTRIBUTE_RUN` to retrieve
`index` - an index within the text

Returns:

an `AccessibleTextSequence` specifying the text if `part` and `index` are valid. Otherwise, `null` is returned

Since:

1.6

See Also:

`AccessibleText.CHARACTER`, `AccessibleText.WORD`, `AccessibleText.SENTENCE`,
`AccessibleExtendedText.LINE`, `AccessibleExtendedText.ATTRIBUTE_RUN`

getTextBounds

```
public Rectangle getTextBounds(int startIndex,  
                                int endIndex)
```

Returns the `Rectangle` enclosing the text between two indices.

Specified by:

`getTextBounds` in interface `AccessibleExtendedText`

Parameters:

startIndex - the start index in the text
endIndex - the end index in the text

Returns:

the bounding rectangle of the text if the indices are valid. Otherwise, null is returned

Since:

1.6

getAccessibleAction

```
public AccessibleAction getAccessibleAction()
```

Overrides:

getAccessibleAction in class AccessibleContext

getAccessibleActionCount

```
public int getAccessibleActionCount()
```

Returns the number of accessible actions available in this object. If there are more than one, the first one is considered the "default" action of the object.

Specified by:

getAccessibleActionCount in interface AccessibleAction

Returns:

the zero-based number of Actions in this object

Since:

1.4

getAccessibleActionDescription

```
public String getAccessibleActionDescription(int i)
```

Returns a description of the specified action of the object.

Specified by:

getAccessibleActionDescription in interface AccessibleAction

Parameters:

i - zero-based index of the actions

Returns:

a String description of the action

Since:

1.4

See Also:

[getAccessibleActionCount\(\)](#)

doAccessibleAction

```
public boolean doAccessibleAction(int i)
```

Performs the specified Action on the object

Specified by:

doAccessibleAction in interface AccessibleAction

Parameters:

i - zero-based index of actions

Returns:

true if the action was performed; otherwise false.

Since:

1.4

See Also:

[getAccessibleActionCount\(\)](#)

javafx.swing.text

JTextComponent.DropLocation

java.lang.Object

└─ javafx.swing.TransferHandler.DropLocation

└─ **javafx.swing.text.JTextComponent.DropLocation****Enclosing class:**[JTextComponent](#)

```
public static final class JTextComponent.DropLocation
    extends TransferHandler.DropLocation
```

Represents a drop location for JTextComponents.

Since:

1.6

See Also:[JTextComponent.getDropLocation\(\)](#)

Method Summary		Page
Position.Bias	getBias() Returns the bias for the drop index.	68
int	getIndex() Returns the index where dropped data should be inserted into the associated component.	68
String	toString() Returns a string representation of this drop location.	69

Method Detail

getIndex

```
public int getIndex()
```

Returns the index where dropped data should be inserted into the associated component. This index represents a position between characters, as would be interpreted by a caret.

Returns:

the drop index

getBias

```
public Position.Bias getBias()
```

Returns the bias for the drop index.

Returns:

the drop bias

toString

```
public String toString()
```

Returns a string representation of this drop location. This method is intended to be used for debugging purposes, and the content and format of the returned string may vary between implementations.

Overrides:

`toString` in class `TransferHandler.DropLocation`

Returns:

a string representation of this drop location

javax.swing.text

JTextComponent.KeyBinding

java.lang.Object

└─ javax.swing.text.JTextComponent.KeyBinding

Enclosing class:

[JTextComponent](#)

```
public static class JTextComponent.KeyBinding
extends Object
```

Binding record for creating key bindings.

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. As of 1.4, support for long term storage of all JavaBeans™ has been added to the `java.beans` package. Please see `XMLEncoder`.

Field Summary		Page
String	actionName The name of the action for the key.	70
KeyStroke	key The key.	70

Constructor Summary		Page
JTextComponent.KeyBinding (KeyStroke key, String actionName) Creates a new key binding.		70

Field Detail

actionName

```
public String actionName
```

The name of the action for the key.

key

```
public KeyStroke key
```

The key.

Constructor Detail

JTextComponent.KeyBinding

```
public JTextComponent.KeyBinding(KeyStroke key,
String actionName)
```

Creates a new key binding.

Parameters:

key - the key

actionName - the name of the action for the key

Java API documentation generated with [DocFlex/Javadoc](#) 1.6.0 using [JavadocPro](#) template set.

[DocFlex/Javadoc](#) is a template-driven programming tool for rapid development of any Javadoc-based Java API documentation generators (i.e. doclets). If you need to customize your Javadoc without writing a full-blown doclet from scratch, [DocFlex/Javadoc](#) may be the only tool able to help you! Find out more at www.docflex.com