| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/MaskFormatter.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/text/LayoutQueue.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/text/MutableAttributeSet.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/text/MaskFormatter.html)    [**NO FRAMES**](http://docs.google.com/MaskFormatter.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#4d34og8) | [METHOD](#3rdcrjn) |

## **javax.swing.text**

Class MaskFormatter

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [javax.swing.JFormattedTextField.AbstractFormatter](http://docs.google.com/javax/swing/JFormattedTextField.AbstractFormatter.html)  
 [javax.swing.text.DefaultFormatter](http://docs.google.com/javax/swing/text/DefaultFormatter.html)  
 **javax.swing.text.MaskFormatter**

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html), [Cloneable](http://docs.google.com/java/lang/Cloneable.html)

public class **MaskFormatter**extends [DefaultFormatter](http://docs.google.com/javax/swing/text/DefaultFormatter.html)

MaskFormatter is used to format and edit strings. The behavior of a MaskFormatter is controlled by way of a String mask that specifies the valid characters that can be contained at a particular location in the Document model. The following characters can be specified:

| Character | Description |
| --- | --- |
| # | Any valid number, uses Character.isDigit. |
| ' | Escape character, used to escape any of the special formatting characters. |
| U | Any character (Character.isLetter). All lowercase letters are mapped to upper case. |
| L | Any character (Character.isLetter). All upper case letters are mapped to lower case. |
| A | Any character or number (Character.isLetter or Character.isDigit) |
| ? | Any character (Character.isLetter). |
| \* | Anything. |
| H | Any hex character (0-9, a-f or A-F). |

Typically characters correspond to one char, but in certain languages this is not the case. The mask is on a per character basis, and will thus adjust to fit as many chars as are needed.

You can further restrict the characters that can be input by the setInvalidCharacters and setValidCharacters methods. setInvalidCharacters allows you to specify which characters are not legal. setValidCharacters allows you to specify which characters are valid. For example, the following code block is equivalent to a mask of '0xHHH' with no invalid/valid characters:

MaskFormatter formatter = new MaskFormatter("0x\*\*\*");  
 formatter.setValidCharacters("0123456789abcdefABCDEF");

When initially formatting a value if the length of the string is less than the length of the mask, two things can happen. Either the placeholder string will be used, or the placeholder character will be used. Precedence is given to the placeholder string. For example:

MaskFormatter formatter = new MaskFormatter("###-####");  
 formatter.setPlaceholderCharacter('\_');  
 formatter.getDisplayValue(tf, "123");

Would result in the string '123-\_\_\_\_'. If setPlaceholder("555-1212") was invoked '123-1212' would result. The placeholder String is only used on the initial format, on subsequent formats only the placeholder character will be used.

If a MaskFormatter is configured to only allow valid characters (setAllowsInvalid(false)) literal characters will be skipped as necessary when editing. Consider a MaskFormatter with the mask "###-####" and current value "555-1212". Using the right arrow key to navigate through the field will result in (| indicates the position of the caret):

|555-1212  
 5|55-1212  
 55|5-1212  
 555-|1212  
 555-1|212

The '-' is a literal (non-editable) character, and is skipped.

Similar behavior will result when editing. Consider inserting the string '123-45' and '12345' into the MaskFormatter in the previous example. Both inserts will result in the same String, '123-45\_\_'. When MaskFormatter is processing the insert at character position 3 (the '-'), two things can happen:

1. If the inserted character is '-', it is accepted.
2. If the inserted character matches the mask for the next non-literal character, it is accepted at the new location.
3. Anything else results in an invalid edit

By default MaskFormatter will not allow invalid edits, you can change this with the setAllowsInvalid method, and will commit edits on valid edits (use the setCommitsOnValidEdit to change this).

By default, MaskFormatter is in overwrite mode. That is as characters are typed a new character is not inserted, rather the character at the current location is replaced with the newly typed character. You can change this behavior by way of the method setOverwriteMode.

**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 JavaBeansTM has been added to the java.beans package. Please see [XMLEncoder](http://docs.google.com/java/beans/XMLEncoder.html).

**Since:** 1.4

| **Constructor Summary** | |
| --- | --- |
| [**MaskFormatter**](http://docs.google.com/javax/swing/text/MaskFormatter.html#MaskFormatter())()            Creates a MaskFormatter with no mask. |
| [**MaskFormatter**](http://docs.google.com/javax/swing/text/MaskFormatter.html#MaskFormatter(java.lang.String))([String](http://docs.google.com/java/lang/String.html) mask)            Creates a MaskFormatter with the specified mask. |

| **Method Summary** | |
| --- | --- |
| [String](http://docs.google.com/java/lang/String.html) | [**getInvalidCharacters**](http://docs.google.com/javax/swing/text/MaskFormatter.html#getInvalidCharacters())()            Returns the characters that are not valid for input. |
| [String](http://docs.google.com/java/lang/String.html) | [**getMask**](http://docs.google.com/javax/swing/text/MaskFormatter.html#getMask())()            Returns the formatting mask. |
| [String](http://docs.google.com/java/lang/String.html) | [**getPlaceholder**](http://docs.google.com/javax/swing/text/MaskFormatter.html#getPlaceholder())()            Returns the String to use if the value does not completely fill in the mask. |
| char | [**getPlaceholderCharacter**](http://docs.google.com/javax/swing/text/MaskFormatter.html#getPlaceholderCharacter())()            Returns the character to use in place of characters that are not present in the value, ie the user must fill them in. |
| [String](http://docs.google.com/java/lang/String.html) | [**getValidCharacters**](http://docs.google.com/javax/swing/text/MaskFormatter.html#getValidCharacters())()            Returns the valid characters that can be input. |
| boolean | [**getValueContainsLiteralCharacters**](http://docs.google.com/javax/swing/text/MaskFormatter.html#getValueContainsLiteralCharacters())()            Returns true if stringToValue should return literal characters in the mask. |
| void | [**install**](http://docs.google.com/javax/swing/text/MaskFormatter.html#install(javax.swing.JFormattedTextField))([JFormattedTextField](http://docs.google.com/javax/swing/JFormattedTextField.html) ftf)            Installs the DefaultFormatter onto a particular JFormattedTextField. |
| void | [**setInvalidCharacters**](http://docs.google.com/javax/swing/text/MaskFormatter.html#setInvalidCharacters(java.lang.String))([String](http://docs.google.com/java/lang/String.html) invalidCharacters)            Allows for further restricting of the characters that can be input. |
| void | [**setMask**](http://docs.google.com/javax/swing/text/MaskFormatter.html#setMask(java.lang.String))([String](http://docs.google.com/java/lang/String.html) mask)            Sets the mask dictating the legal characters. |
| void | [**setPlaceholder**](http://docs.google.com/javax/swing/text/MaskFormatter.html#setPlaceholder(java.lang.String))([String](http://docs.google.com/java/lang/String.html) placeholder)            Sets the string to use if the value does not completely fill in the mask. |
| void | [**setPlaceholderCharacter**](http://docs.google.com/javax/swing/text/MaskFormatter.html#setPlaceholderCharacter(char))(char placeholder)            Sets the character to use in place of characters that are not present in the value, ie the user must fill them in. |
| void | [**setValidCharacters**](http://docs.google.com/javax/swing/text/MaskFormatter.html#setValidCharacters(java.lang.String))([String](http://docs.google.com/java/lang/String.html) validCharacters)            Allows for further restricting of the characters that can be input. |
| void | [**setValueContainsLiteralCharacters**](http://docs.google.com/javax/swing/text/MaskFormatter.html#setValueContainsLiteralCharacters(boolean))(boolean containsLiteralChars)            If true, the returned value and set value will also contain the literal characters in mask. |
| [Object](http://docs.google.com/java/lang/Object.html) | [**stringToValue**](http://docs.google.com/javax/swing/text/MaskFormatter.html#stringToValue(java.lang.String))([String](http://docs.google.com/java/lang/String.html) value)            Parses the text, returning the appropriate Object representation of the String value. |
| [String](http://docs.google.com/java/lang/String.html) | [**valueToString**](http://docs.google.com/javax/swing/text/MaskFormatter.html#valueToString(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) value)            Returns a String representation of the Object value based on the mask. |

| **Methods inherited from class javax.swing.text.**[**DefaultFormatter**](http://docs.google.com/javax/swing/text/DefaultFormatter.html) |
| --- |
| [clone](http://docs.google.com/javax/swing/text/DefaultFormatter.html#clone()), [getAllowsInvalid](http://docs.google.com/javax/swing/text/DefaultFormatter.html#getAllowsInvalid()), [getCommitsOnValidEdit](http://docs.google.com/javax/swing/text/DefaultFormatter.html#getCommitsOnValidEdit()), [getDocumentFilter](http://docs.google.com/javax/swing/text/DefaultFormatter.html#getDocumentFilter()), [getNavigationFilter](http://docs.google.com/javax/swing/text/DefaultFormatter.html#getNavigationFilter()), [getOverwriteMode](http://docs.google.com/javax/swing/text/DefaultFormatter.html#getOverwriteMode()), [getValueClass](http://docs.google.com/javax/swing/text/DefaultFormatter.html#getValueClass()), [setAllowsInvalid](http://docs.google.com/javax/swing/text/DefaultFormatter.html#setAllowsInvalid(boolean)), [setCommitsOnValidEdit](http://docs.google.com/javax/swing/text/DefaultFormatter.html#setCommitsOnValidEdit(boolean)), [setOverwriteMode](http://docs.google.com/javax/swing/text/DefaultFormatter.html#setOverwriteMode(boolean)), [setValueClass](http://docs.google.com/javax/swing/text/DefaultFormatter.html#setValueClass(java.lang.Class)) |

| **Methods inherited from class javax.swing.**[**JFormattedTextField.AbstractFormatter**](http://docs.google.com/javax/swing/JFormattedTextField.AbstractFormatter.html) |
| --- |
| [getActions](http://docs.google.com/javax/swing/JFormattedTextField.AbstractFormatter.html#getActions()), [getFormattedTextField](http://docs.google.com/javax/swing/JFormattedTextField.AbstractFormatter.html#getFormattedTextField()), [invalidEdit](http://docs.google.com/javax/swing/JFormattedTextField.AbstractFormatter.html#invalidEdit()), [setEditValid](http://docs.google.com/javax/swing/JFormattedTextField.AbstractFormatter.html#setEditValid(boolean)), [uninstall](http://docs.google.com/javax/swing/JFormattedTextField.AbstractFormatter.html#uninstall()) |

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

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

### MaskFormatter

public **MaskFormatter**()

Creates a MaskFormatter with no mask.

### MaskFormatter

public **MaskFormatter**([String](http://docs.google.com/java/lang/String.html) mask)  
 throws [ParseException](http://docs.google.com/java/text/ParseException.html)

Creates a MaskFormatter with the specified mask. A ParseException will be thrown if mask is an invalid mask.

**Throws:** [ParseException](http://docs.google.com/java/text/ParseException.html) - if mask does not contain valid mask characters

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

### setMask

public void **setMask**([String](http://docs.google.com/java/lang/String.html) mask)  
 throws [ParseException](http://docs.google.com/java/text/ParseException.html)

Sets the mask dictating the legal characters. This will throw a ParseException if mask is not valid.

**Throws:** [ParseException](http://docs.google.com/java/text/ParseException.html) - if mask does not contain valid mask characters

### getMask

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

Returns the formatting mask.

**Returns:**Mask dictating legal character values.

### setValidCharacters

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

Allows for further restricting of the characters that can be input. Only characters specified in the mask, not in the invalidCharacters, and in validCharacters will be allowed to be input. Passing in null (the default) implies the valid characters are only bound by the mask and the invalid characters.

**Parameters:**validCharacters - If non-null, specifies legal characters.

### getValidCharacters

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

Returns the valid characters that can be input.

**Returns:**Legal characters

### setInvalidCharacters

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

Allows for further restricting of the characters that can be input. Only characters specified in the mask, not in the invalidCharacters, and in validCharacters will be allowed to be input. Passing in null (the default) implies the valid characters are only bound by the mask and the valid characters.

**Parameters:**invalidCharacters - If non-null, specifies illegal characters.

### getInvalidCharacters

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

Returns the characters that are not valid for input.

**Returns:**illegal characters.

### setPlaceholder

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

Sets the string to use if the value does not completely fill in the mask. A null value implies the placeholder char should be used.

**Parameters:**placeholder - String used when formatting if the value does not completely fill the mask

### getPlaceholder

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

Returns the String to use if the value does not completely fill in the mask.

**Returns:**String used when formatting if the value does not completely fill the mask

### setPlaceholderCharacter

public void **setPlaceholderCharacter**(char placeholder)

Sets the character to use in place of characters that are not present in the value, ie the user must fill them in. The default value is a space.

This is only applicable if the placeholder string has not been specified, or does not completely fill in the mask.

**Parameters:**placeholder - Character used when formatting if the value does not completely fill the mask

### getPlaceholderCharacter

public char **getPlaceholderCharacter**()

Returns the character to use in place of characters that are not present in the value, ie the user must fill them in.

**Returns:**Character used when formatting if the value does not completely fill the mask

### setValueContainsLiteralCharacters

public void **setValueContainsLiteralCharacters**(boolean containsLiteralChars)

If true, the returned value and set value will also contain the literal characters in mask.

For example, if the mask is '(###) ###-####', the current value is '(415) 555-1212', and valueContainsLiteralCharacters is true stringToValue will return '(415) 555-1212'. On the other hand, if valueContainsLiteralCharacters is false, stringToValue will return '4155551212'.

**Parameters:**containsLiteralChars - Used to indicate if literal characters in mask should be returned in stringToValue

### getValueContainsLiteralCharacters

public boolean **getValueContainsLiteralCharacters**()

Returns true if stringToValue should return literal characters in the mask.

**Returns:**True if literal characters in mask should be returned in stringToValue

### stringToValue

public [Object](http://docs.google.com/java/lang/Object.html) **stringToValue**([String](http://docs.google.com/java/lang/String.html) value)  
 throws [ParseException](http://docs.google.com/java/text/ParseException.html)

Parses the text, returning the appropriate Object representation of the String value. This strips the literal characters as necessary and invokes supers stringToValue, so that if you have specified a value class (setValueClass) an instance of it will be created. This will throw a ParseException if the value does not match the current mask. Refer to [setValueContainsLiteralCharacters(boolean)](http://docs.google.com/javax/swing/text/MaskFormatter.html#setValueContainsLiteralCharacters(boolean)) for details on how literals are treated.

**Overrides:**[stringToValue](http://docs.google.com/javax/swing/text/DefaultFormatter.html#stringToValue(java.lang.String)) in class [DefaultFormatter](http://docs.google.com/javax/swing/text/DefaultFormatter.html) **Parameters:**value - String to convert **Returns:**Object representation of text **Throws:** [ParseException](http://docs.google.com/java/text/ParseException.html) - if there is an error in the conversion**See Also:**[setValueContainsLiteralCharacters(boolean)](http://docs.google.com/javax/swing/text/MaskFormatter.html#setValueContainsLiteralCharacters(boolean))

### valueToString

public [String](http://docs.google.com/java/lang/String.html) **valueToString**([Object](http://docs.google.com/java/lang/Object.html) value)  
 throws [ParseException](http://docs.google.com/java/text/ParseException.html)

Returns a String representation of the Object value based on the mask. Refer to [setValueContainsLiteralCharacters(boolean)](http://docs.google.com/javax/swing/text/MaskFormatter.html#setValueContainsLiteralCharacters(boolean)) for details on how literals are treated.

**Overrides:**[valueToString](http://docs.google.com/javax/swing/text/DefaultFormatter.html#valueToString(java.lang.Object)) in class [DefaultFormatter](http://docs.google.com/javax/swing/text/DefaultFormatter.html) **Parameters:**value - Value to convert **Returns:**String representation of value **Throws:** [ParseException](http://docs.google.com/java/text/ParseException.html) - if there is an error in the conversion**See Also:**[setValueContainsLiteralCharacters(boolean)](http://docs.google.com/javax/swing/text/MaskFormatter.html#setValueContainsLiteralCharacters(boolean))

### install

public void **install**([JFormattedTextField](http://docs.google.com/javax/swing/JFormattedTextField.html) ftf)

Installs the DefaultFormatter onto a particular JFormattedTextField. This will invoke valueToString to convert the current value from the JFormattedTextField to a String. This will then install the Actions from getActions, the DocumentFilter returned from getDocumentFilter and the NavigationFilter returned from getNavigationFilter onto the JFormattedTextField.

Subclasses will typically only need to override this if they wish to install additional listeners on the JFormattedTextField.

If there is a ParseException in converting the current value to a String, this will set the text to an empty String, and mark the JFormattedTextField as being in an invalid state.

While this is a public method, this is typically only useful for subclassers of JFormattedTextField. JFormattedTextField will invoke this method at the appropriate times when the value changes, or its internal state changes.

**Overrides:**[install](http://docs.google.com/javax/swing/text/DefaultFormatter.html#install(javax.swing.JFormattedTextField)) in class [DefaultFormatter](http://docs.google.com/javax/swing/text/DefaultFormatter.html) **Parameters:**ftf - JFormattedTextField to format for, may be null indicating uninstall from current JFormattedTextField.

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/MaskFormatter.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/swing/text/LayoutQueue.html)   [**NEXT CLASS**](http://docs.google.com/javax/swing/text/MutableAttributeSet.html) | [**FRAMES**](http://docs.google.com/index.html?javax/swing/text/MaskFormatter.html)    [**NO FRAMES**](http://docs.google.com/MaskFormatter.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#4d34og8) | [METHOD](#3rdcrjn) |

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

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

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