| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/EventHandler.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/java/beans/Encoder.html)   [**NEXT CLASS**](http://docs.google.com/java/beans/EventSetDescriptor.html) | [**FRAMES**](http://docs.google.com/index.html?java/beans/EventHandler.html)    [**NO FRAMES**](http://docs.google.com/EventHandler.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#4d34og8) |

## **java.beans**

Class EventHandler

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

**All Implemented Interfaces:** [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html)

public class **EventHandler**extends [Object](http://docs.google.com/java/lang/Object.html)implements [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html)

The EventHandler class provides support for dynamically generating event listeners whose methods execute a simple statement involving an incoming event object and a target object.

The EventHandler class is intended to be used by interactive tools, such as application builders, that allow developers to make connections between beans. Typically connections are made from a user interface bean (the event *source*) to an application logic bean (the *target*). The most effective connections of this kind isolate the application logic from the user interface. For example, the EventHandler for a connection from a JCheckBox to a method that accepts a boolean value can deal with extracting the state of the check box and passing it directly to the method so that the method is isolated from the user interface layer.

Inner classes are another, more general way to handle events from user interfaces. The EventHandler class handles only a subset of what is possible using inner classes. However, EventHandler works better with the long-term persistence scheme than inner classes. Also, using EventHandler in large applications in which the same interface is implemented many times can reduce the disk and memory footprint of the application.

The reason that listeners created with EventHandler have such a small footprint is that the Proxy class, on which the EventHandler relies, shares implementations of identical interfaces. For example, if you use the EventHandler create methods to make all the ActionListeners in an application, all the action listeners will be instances of a single class (one created by the Proxy class). In general, listeners based on the Proxy class require one listener class to be created per *listener type* (interface), whereas the inner class approach requires one class to be created per *listener* (object that implements the interface).

You don't generally deal directly with EventHandler instances. Instead, you use one of the EventHandler create methods to create an object that implements a given listener interface. This listener object uses an EventHandler object behind the scenes to encapsulate information about the event, the object to be sent a message when the event occurs, the message (method) to be sent, and any argument to the method. The following section gives examples of how to create listener objects using the create methods.

## Examples of Using EventHandler

The simplest use of EventHandler is to install a listener that calls a method on the target object with no arguments. In the following example we create an ActionListener that invokes the toFront method on an instance of javax.swing.JFrame.

myButton.addActionListener(  
 (ActionListener)EventHandler.create(ActionListener.class, frame, "toFront"));

When myButton is pressed, the statement frame.toFront() will be executed. One could get the same effect, with some additional compile-time type safety, by defining a new implementation of the ActionListener interface and adding an instance of it to the button:

//Equivalent code using an inner class instead of EventHandler.  
myButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 frame.toFront();  
 }  
});

The next simplest use of EventHandler is to extract a property value from the first argument of the method in the listener interface (typically an event object) and use it to set the value of a property in the target object. In the following example we create an ActionListener that sets the nextFocusableComponent property of the target (myButton) object to the value of the "source" property of the event.

EventHandler.create(ActionListener.class, myButton, "nextFocusableComponent", "source")

This would correspond to the following inner class implementation:

//Equivalent code using an inner class instead of EventHandler.  
new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 myButton.setNextFocusableComponent((Component)e.getSource());   
 }  
}

It's also possible to create an EventHandler that just passes the incoming event object to the target's action. If the fourth EventHandler.create argument is an empty string, then the event is just passed along:

EventHandler.create(ActionListener.class, target, "doActionEvent", "")

This would correspond to the following inner class implementation:

//Equivalent code using an inner class instead of EventHandler.  
new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 target.doActionEvent(e);  
 }  
}

Probably the most common use of EventHandler is to extract a property value from the *source* of the event object and set this value as the value of a property of the target object. In the following example we create an ActionListener that sets the "label" property of the target object to the value of the "text" property of the source (the value of the "source" property) of the event.

EventHandler.create(ActionListener.class, myButton, "label", "source.text")

This would correspond to the following inner class implementation:

//Equivalent code using an inner class instead of EventHandler.  
new ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 myButton.setLabel(((JTextField)e.getSource()).getText());   
 }  
}

The event property may be "qualified" with an arbitrary number of property prefixes delimited with the "." character. The "qualifying" names that appear before the "." characters are taken as the names of properties that should be applied, left-most first, to the event object.

For example, the following action listener

EventHandler.create(ActionListener.class, target, "a", "b.c.d")

might be written as the following inner class (assuming all the properties had canonical getter methods and returned the appropriate types):

//Equivalent code using an inner class instead of EventHandler.  
new ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 target.setA(e.getB().getC().isD());   
 }  
}

The target property may also be "qualified" with an arbitrary number of property prefixs delimited with the "." character. For example, the following action listener:

EventHandler.create(ActionListener.class, target, "a.b", "c.d")

might be written as the following inner class (assuming all the properties had canonical getter methods and returned the appropriate types):

//Equivalent code using an inner class instead of EventHandler.  
 new ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 target.getA().setB(e.getC().isD());   
 }  
}

As EventHandler ultimately relies on reflection to invoke a method we recommend against targeting an overloaded method. For example, if the target is an instance of the class MyTarget which is defined as:

public class MyTarget {  
 public void doIt(String);  
 public void doIt(Object);  
 }

Then the method doIt is overloaded. EventHandler will invoke the method that is appropriate based on the source. If the source is null, then either method is appropriate and the one that is invoked is undefined. For that reason we recommend against targeting overloaded methods.

**Since:** 1.4 **See Also:**[Proxy](http://docs.google.com/java/lang/reflect/Proxy.html), [EventObject](http://docs.google.com/java/util/EventObject.html)

| **Constructor Summary** | |
| --- | --- |
| [**EventHandler**](http://docs.google.com/java/beans/EventHandler.html#EventHandler(java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String))([Object](http://docs.google.com/java/lang/Object.html) target, [String](http://docs.google.com/java/lang/String.html) action, [String](http://docs.google.com/java/lang/String.html) eventPropertyName, [String](http://docs.google.com/java/lang/String.html) listenerMethodName)            Creates a new EventHandler object; you generally use one of the create methods instead of invoking this constructor directly. |

| **Method Summary** | |
| --- | --- |
| static   | <T> T | | --- | | [**create**](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String))([Class](http://docs.google.com/java/lang/Class.html)<T> listenerInterface, [Object](http://docs.google.com/java/lang/Object.html) target, [String](http://docs.google.com/java/lang/String.html) action)            Creates an implementation of listenerInterface in which *all* of the methods in the listener interface apply the handler's action to the target. |
| static   | <T> T | | --- | | [**create**](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String))([Class](http://docs.google.com/java/lang/Class.html)<T> listenerInterface, [Object](http://docs.google.com/java/lang/Object.html) target, [String](http://docs.google.com/java/lang/String.html) action, [String](http://docs.google.com/java/lang/String.html) eventPropertyName)            /\*\* Creates an implementation of listenerInterface in which *all* of the methods pass the value of the event expression, eventPropertyName, to the final method in the statement, action, which is applied to the target. |
| static   | <T> T | | --- | | [**create**](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String))([Class](http://docs.google.com/java/lang/Class.html)<T> listenerInterface, [Object](http://docs.google.com/java/lang/Object.html) target, [String](http://docs.google.com/java/lang/String.html) action, [String](http://docs.google.com/java/lang/String.html) eventPropertyName, [String](http://docs.google.com/java/lang/String.html) listenerMethodName)            Creates an implementation of listenerInterface in which the method named listenerMethodName passes the value of the event expression, eventPropertyName, to the final method in the statement, action, which is applied to the target. |
| [String](http://docs.google.com/java/lang/String.html) | [**getAction**](http://docs.google.com/java/beans/EventHandler.html#getAction())()            Returns the name of the target's writable property that this event handler will set, or the name of the method that this event handler will invoke on the target. |
| [String](http://docs.google.com/java/lang/String.html) | [**getEventPropertyName**](http://docs.google.com/java/beans/EventHandler.html#getEventPropertyName())()            Returns the property of the event that should be used in the action applied to the target. |
| [String](http://docs.google.com/java/lang/String.html) | [**getListenerMethodName**](http://docs.google.com/java/beans/EventHandler.html#getListenerMethodName())()            Returns the name of the method that will trigger the action. |
| [Object](http://docs.google.com/java/lang/Object.html) | [**getTarget**](http://docs.google.com/java/beans/EventHandler.html#getTarget())()            Returns the object to which this event handler will send a message. |
| [Object](http://docs.google.com/java/lang/Object.html) | [**invoke**](http://docs.google.com/java/beans/EventHandler.html#invoke(java.lang.Object,%20java.lang.reflect.Method,%20java.lang.Object%5B%5D))([Object](http://docs.google.com/java/lang/Object.html) proxy, [Method](http://docs.google.com/java/lang/reflect/Method.html) method, [Object](http://docs.google.com/java/lang/Object.html)[] arguments)            Extract the appropriate property value from the event and pass it to the action associated with this EventHandler. |

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

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

### EventHandler

public **EventHandler**([Object](http://docs.google.com/java/lang/Object.html) target,  
 [String](http://docs.google.com/java/lang/String.html) action,  
 [String](http://docs.google.com/java/lang/String.html) eventPropertyName,  
 [String](http://docs.google.com/java/lang/String.html) listenerMethodName)

Creates a new EventHandler object; you generally use one of the create methods instead of invoking this constructor directly. Refer to [the general version of create](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String)) for a complete description of the eventPropertyName and listenerMethodName parameter.

**Parameters:**target - the object that will perform the actionaction - the name of a (possibly qualified) property or method on the targeteventPropertyName - the (possibly qualified) name of a readable property of the incoming eventlistenerMethodName - the name of the method in the listener interface that should trigger the action **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if target is null [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if action is null**See Also:**[EventHandler](http://docs.google.com/java/beans/EventHandler.html), [create(Class, Object, String, String, String)](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String)), [getTarget()](http://docs.google.com/java/beans/EventHandler.html#getTarget()), [getAction()](http://docs.google.com/java/beans/EventHandler.html#getAction()), [getEventPropertyName()](http://docs.google.com/java/beans/EventHandler.html#getEventPropertyName()), [getListenerMethodName()](http://docs.google.com/java/beans/EventHandler.html#getListenerMethodName())

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

### getTarget

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

Returns the object to which this event handler will send a message.

**Returns:**the target of this event handler**See Also:**[EventHandler(Object, String, String, String)](http://docs.google.com/java/beans/EventHandler.html#EventHandler(java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String))

### getAction

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

Returns the name of the target's writable property that this event handler will set, or the name of the method that this event handler will invoke on the target.

**Returns:**the action of this event handler**See Also:**[EventHandler(Object, String, String, String)](http://docs.google.com/java/beans/EventHandler.html#EventHandler(java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String))

### getEventPropertyName

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

Returns the property of the event that should be used in the action applied to the target.

**Returns:**the property of the event**See Also:**[EventHandler(Object, String, String, String)](http://docs.google.com/java/beans/EventHandler.html#EventHandler(java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String))

### getListenerMethodName

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

Returns the name of the method that will trigger the action. A return value of null signifies that all methods in the listener interface trigger the action.

**Returns:**the name of the method that will trigger the action**See Also:**[EventHandler(Object, String, String, String)](http://docs.google.com/java/beans/EventHandler.html#EventHandler(java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String))

### invoke

public [Object](http://docs.google.com/java/lang/Object.html) **invoke**([Object](http://docs.google.com/java/lang/Object.html) proxy,  
 [Method](http://docs.google.com/java/lang/reflect/Method.html) method,  
 [Object](http://docs.google.com/java/lang/Object.html)[] arguments)

Extract the appropriate property value from the event and pass it to the action associated with this EventHandler.

**Specified by:**[invoke](http://docs.google.com/java/lang/reflect/InvocationHandler.html#invoke(java.lang.Object,%20java.lang.reflect.Method,%20java.lang.Object%5B%5D)) in interface [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) **Parameters:**proxy - the proxy objectmethod - the method in the listener interfacearguments - an array of objects containing the values of the arguments passed in the method invocation on the proxy instance, or null if interface method takes no arguments. Arguments of primitive types are wrapped in instances of the appropriate primitive wrapper class, such as java.lang.Integer or java.lang.Boolean. **Returns:**the result of applying the action to the target**See Also:**[EventHandler](http://docs.google.com/java/beans/EventHandler.html)

### create

public static <T> T **create**([Class](http://docs.google.com/java/lang/Class.html)<T> listenerInterface,  
 [Object](http://docs.google.com/java/lang/Object.html) target,  
 [String](http://docs.google.com/java/lang/String.html) action)

Creates an implementation of listenerInterface in which *all* of the methods in the listener interface apply the handler's action to the target. This method is implemented by calling the other, more general, implementation of the create method with both the eventPropertyName and the listenerMethodName taking the value null. Refer to [the general version of create](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String)) for a complete description of the action parameter.

To create an ActionListener that shows a JDialog with dialog.show(), one can write:

EventHandler.create(ActionListener.class, dialog, "show")

**Parameters:**listenerInterface - the listener interface to create a proxy fortarget - the object that will perform the actionaction - the name of a (possibly qualified) property or method on the target **Returns:**an object that implements listenerInterface **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if listenerInterface is null [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if target is null [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if action is null**See Also:**[create(Class, Object, String, String)](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String))

### create

public static <T> T **create**([Class](http://docs.google.com/java/lang/Class.html)<T> listenerInterface,  
 [Object](http://docs.google.com/java/lang/Object.html) target,  
 [String](http://docs.google.com/java/lang/String.html) action,  
 [String](http://docs.google.com/java/lang/String.html) eventPropertyName)

/\*\* Creates an implementation of listenerInterface in which *all* of the methods pass the value of the event expression, eventPropertyName, to the final method in the statement, action, which is applied to the target. This method is implemented by calling the more general, implementation of the create method with the listenerMethodName taking the value null. Refer to [the general version of create](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String)) for a complete description of the action and eventPropertyName parameters.

To create an ActionListener that sets the the text of a JLabel to the text value of the JTextField source of the incoming event, you can use the following code:

EventHandler.create(ActionListener.class, label, "text", "source.text");

This is equivalent to the following code:

//Equivalent code using an inner class instead of EventHandler.  
new ActionListener() {  
 public void actionPerformed(ActionEvent event) {  
 label.setText(((JTextField)(event.getSource())).getText());  
 }  
};

**Parameters:**listenerInterface - the listener interface to create a proxy fortarget - the object that will perform the actionaction - the name of a (possibly qualified) property or method on the targeteventPropertyName - the (possibly qualified) name of a readable property of the incoming event **Returns:**an object that implements listenerInterface **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if listenerInterface is null [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if target is null [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if action is null**See Also:**[create(Class, Object, String, String, String)](http://docs.google.com/java/beans/EventHandler.html#create(java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String))

### create

public static <T> T **create**([Class](http://docs.google.com/java/lang/Class.html)<T> listenerInterface,  
 [Object](http://docs.google.com/java/lang/Object.html) target,  
 [String](http://docs.google.com/java/lang/String.html) action,  
 [String](http://docs.google.com/java/lang/String.html) eventPropertyName,  
 [String](http://docs.google.com/java/lang/String.html) listenerMethodName)

Creates an implementation of listenerInterface in which the method named listenerMethodName passes the value of the event expression, eventPropertyName, to the final method in the statement, action, which is applied to the target. All of the other listener methods do nothing.

The eventPropertyName string is used to extract a value from the incoming event object that is passed to the target method. The common case is the target method takes no arguments, in which case a value of null should be used for the eventPropertyName. Alternatively if you want the incoming event object passed directly to the target method use the empty string. The format of the eventPropertyName string is a sequence of methods or properties where each method or property is applied to the value returned by the preceeding method starting from the incoming event object. The syntax is: propertyName{.propertyName}\* where propertyName matches a method or property. For example, to extract the point property from a MouseEvent, you could use either "point" or "getPoint" as the eventPropertyName. To extract the "text" property from a MouseEvent with a JLabel source use any of the following as eventPropertyName: "source.text", "getSource.text" "getSource.getText" or "source.getText". If a method can not be found, or an exception is generated as part of invoking a method a RuntimeException will be thrown at dispatch time. For example, if the incoming event object is null, and eventPropertyName is non-null and not empty, a RuntimeException will be thrown.

The action argument is of the same format as the eventPropertyName argument where the last property name identifies either a method name or writable property.

If the listenerMethodName is null *all* methods in the interface trigger the action to be executed on the target.

For example, to create a MouseListener that sets the target object's origin property to the incoming MouseEvent's location (that's the value of mouseEvent.getPoint()) each time a mouse button is pressed, one would write:

EventHandler.create(MouseListener.class, "mousePressed", target, "origin", "point");

This is comparable to writing a MouseListener in which all of the methods except mousePressed are no-ops:

//Equivalent code using an inner class instead of EventHandler.  
new MouseAdapter() {  
 public void mousePressed(MouseEvent e) {  
 target.setOrigin(e.getPoint());  
 }  
};

**Parameters:**listenerInterface - the listener interface to create a proxy fortarget - the object that will perform the actionaction - the name of a (possibly qualified) property or method on the targeteventPropertyName - the (possibly qualified) name of a readable property of the incoming eventlistenerMethodName - the name of the method in the listener interface that should trigger the action **Returns:**an object that implements listenerInterface **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if listenerInterface is null [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if target is null [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if action is null**See Also:**[EventHandler](http://docs.google.com/java/beans/EventHandler.html)

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/EventHandler.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/java/beans/Encoder.html)   [**NEXT CLASS**](http://docs.google.com/java/beans/EventSetDescriptor.html) | [**FRAMES**](http://docs.google.com/index.html?java/beans/EventHandler.html)    [**NO FRAMES**](http://docs.google.com/EventHandler.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#4d34og8) |

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

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

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