# SubmitOnEvent

## Submit on Event

This tomahawk sandbox component allows you trigger a form submit on specific events.

Syntax:

```
<s:submitOnEvent
    for="button_id|link_id"
    event="keypress|keydown|keyup|change|focus|blur|click|mousedown|mouseup|mousemove|mouseover|mouseout|select"
    callback="user_defined_javascript_function" />
```

- *for* has to be a pointer to a commandLink or commandButton.
- *event* is optional and can be one of the strings listed above. Default: **keypress**
- *callback* is optional and can be any user definde javascript function. Default: internal function which triggers "on enter"

The signature for the callback function is

```
function user_defined_javascript_function(event, srcComponentId, clickComponentId)
```

- regardless of the browser you use, you'll have access to an event object through the parameter *event*
- return true if the form submit should happen, else return false

The component can work in three different modes:

1. as child of an input component (inputText, ...)
2. as child of commandLink or commandButton
3. as a "standalone" component somewhere on the page

## as child of an input component (inputText, ...)

In this mode the argument *for* is mandatory. The component will only listen to events of the parent component. This allows you to trigger different actions depending on the source component.

Examples:

Trigger "on enter"

```
    <h:commandButton id="submitButton" value="submit button" action="#{.....buttonAction}"/>

    <h:inputText id="text1" value="....">
        <s:submitOnEvent for="submitButton" />
    </h:inputText>
```

Trigger "on change"

```
    <h:commandButton id="submitButton" value="submit button" action="#{.....buttonAction}"/>

    <h:selectOneMenu id="text3" value="....">
        <f:selectItem itemLabel="selection1" itemValue="selection1" />
        <f:selectItem itemLabel="selection2" itemValue="selection2" />
        <f:selectItem itemLabel="selection3" itemValue="selection3" />
        <f:selectItem itemLabel="selection4" itemValue="selection4" />

        <s:submitOnEvent for="submitButton" event="change" />
    </h:selectOneMenu>
```

Trigger "on user defined"

```
function mySpecialUserCallback(event, srcComponentId, clickComponentId)
{
    var keycode;
    if (event.keyCode)
    {
        keycode = event.keyCode;
    }
    else if (event)
    {
        keycode = event.which;
    }

    return (keycode == 97 || keycode == 65);
}

    <h:commandButton id="submitButton" value="submit button" action="#{.....buttonAction}"/>

    <h:inputText id="text5" value="#{submitOnEvent.strings.text5}">
        <s:submitOnEvent for="submitLink" callback="mySpecialUserCallback"/>
    </h:inputText>
```

## as child of commandLink or commandButton

In this mode the argument *for* has no meaning. The component will trigger the parent link or button by listening to the global event loop. This mode is meant to define a page wide "default action".

Examples:

```
    <h:commandLink id="submitLink" value="submit link" action="#{.....linkAction}">
        <s:submitOnEvent/>
    </h:commandLink>
```

## as a "standalone" component somewhere on the page

In this mode the argument *for* is mandatory. The component will listen to the global event loop and trigger the button pointed to with *for*. Notice: The difference to "mode 2" is, that the *for* argument can be a value binding, so you can trigger different actions depending on the page state.

Examples:

```
    <h:commandLink id="submitLink" value="submit link" action="#{.....linkAction}"/>

    <s:submitOnEvent for="submitLink" />
```